Using the XMLConnector component to connect to external XML files

You can use the XMLConnector component to connect to an external XML document in order to bind to properties in the document. Its purpose is to read or write XML documents using HTTP `GET` operations, `POST` operations, or both. It acts as a connector between other components and external XML documents. The XMLConnector communicates with components in your application using either data binding features in the Flash Professional authoring environment, or ActionScript code. For more information, see "XMLConnector component" in Using Components Help.

Using a Western keyboard to enter Asian characters on the Stage

With Flash MX 2004, you can enter Asian characters on the Stage using a standard Western keyboard by using Input Method Editors (IMEs). Flash supports more than two dozen different IMEs.

For example, if you want to create a website that will reach a broad range of Asian viewers, you can use a standard Western keyboard to create text in Chinese, Japanese, and Korean simply by changing the input method editor.

In previous versions of Flash, it was not possible to input Korean characters using a standard Western keyboard. With Flash MX 2004, you can enter text in Korean, Japanese, and Chinese characters simply by toggling the IME from Japanese and Chinese character input to Korean character input.

Note: This affects only text input on the Stage, not text entered in the Actions panel. This feature is available for all supported Windows operating systems and Macintosh OSX.

To toggle between Japanese and Chinese character input and Korean character input:

1. Select Edit > Preferences and click the Editing tab in the Preferences dialog box.
2. Under Input Language Settings, select one of the following options:
   - Select Chinese and Japanese to input Chinese and Japanese characters from a Western keyboard. (This is the default setting, and it should be selected for Western languages as well.)
   - Select Korean to input Korean characters from a Western keyboard.
3. Click OK.

Using ActionScript to load external files

If you have existing XML data you want to load, or prefer a different format for the XML file, instead of using the Strings panel, you can create a document containing multilanguage text by placing the text in an external text or XML file and loading the file into the movie at runtime, using the `loadVariables` action, the `getURL` action, the `LoadVars` object, or the `XML` object.

You should save the external file in UTF 8 (recommended), UTF 16BE, or UTF 16LE format, using an application that supports the format. If you are using UTF-16BE or UTF-16LE format, the file must begin with a byte order mark (BOM) to identify the encoding format to Flash Player See "Unicode encoding formats supported by Flash Player" on page 217.

Note: If the external file is an XML file, you cannot use an XML encoding tag to change the encoding of the file. You should save the file in a supported Unicode format. See "About encoding in external XML files" on page 217.

To include multilanguage text using an externally loaded file:

1 In the Flash authoring tool, create a dynamic or input text field to display the text in the document. For more information, see Chapter 6, "Working with Text," on page 95.

2 In the Property inspector, with the text field selected, assign an instance name to the text field.

3 Create a text or XML file that defines the value for the text filed variable.

4 Save the file in UTF-8 (recommended), UTF-16BE, or UTF-16LE format.

If you are using UTF-16 format, make sure a byte order mark is included at the beginning of the file to identify the encoding:

- For UTF-16BE, the first byte of the file should be OxFE, and the second byte should be OxFF.
- For UTF-16LE, the first byte of the file should be OxFF, and the second byte should be OxFE.

Note: Most text editors that can save files in UTF-16BE or LE automatically add the BOMs to the files.

5 Use one of the following ActionScript procedures to reference the external file and load it into the dynamic or input text field:

- Use the `loadVariables` action to load an external file. For more information, see `loadVariables()` in ActionScript Dictionary Help.
- Use the `getURL` action to load an external file from a specified URL. For more information, see `getURL()` in ActionScript Dictionary Help.
- Use the `LoadVars` object (a predefined client-server object) to load an external text file from a specified URL. For more information, see `LoadVars class` in ActionScript Dictionary Help.
- Use the `XML` object (a predefined client-server object) to load an external XML file from a specified URL. For more information, see `XML class` in ActionScript Dictionary Help.

Creating documents with multilanguage text using the #include action

You can create a document that contains multiple languages using the `#include` action.

You should save the text file in UTF-8 format. Save the file using an application that supports UTF-8 encoding, such as Dreamweaver.

You must include the following header as the first line of the file, to identify the file as Unicode to the Flash authoring tool:

```
//!-- UTF8
```

Note: Be sure to include a space after the second dash (-).

By default, the Flash authoring application assumes that external files that use the `#include` action are encoded in the traditional code page of the operating system running the authoring tool. Using the `//!-- UTF8` header in a file tells the authoring tool that the external file is encoded as UTF-8.

To include multilanguage text using the #include action:

1 In the Flash authoring tool, create a dynamic or input text field to display the text in the document. For more information, see Chapter 6, "Working with Text," on page 95.

2 In the Property inspector, with the text field selected, assign an instance name to the text field.

3 Create a text file that defines the value for the text field variable. Remember to add the header `//!-- UTF8` at the beginning of the file.

4 Save the file in UTF-8 format.

5 Use the `#include` action to include the external file in the dynamic or input text field. For more information, see `#include` in ActionScript Dictionary Help.

Creating documents with multilanguage text using text variables

You can include Unicode-encoded contents in text variables using the syntax `\uXXXX`, where `XXXX` is the four-digit hexadecimal code point, or escape character, for the Unicode character. The Flash authoring tool supports Unicode escape characters up through `\uFFFF`. To find the code points for Unicode characters, refer to the Unicode Standard at www.Unicode.org.

You can use Unicode escape characters only in text field variables. You cannot include Unicode escape characters in external text or XML files; Flash Player 6 does not recognize Unicode escape characters in external files.

For example, to set a dynamic text field (with the variable name `myTextVar`) that contains Japanese, Korean, Chinese, English, Hebrew, and Greek characters and the euro sign, you can enter the following:

```
myTextVar = "\u304B\uD55C\u6C49hello\u05E2\u03BB\u20AC";
```

When the SWF file plays, the following is displayed in the text field:

か한汉helloעλ€

For best results when creating a text field that contains multiple languages, make sure to use a font that includes all the glyphs needed for your text. See "Using external text or XML files that are not Unicode encoded" on page 230.

Using external text or XML files that are not Unicode encoded

If you load external files into a Flash Player 7 application that are not Unicode-encoded, the text in the external files will not be displayed correctly when Flash Player attempts to display them as Unicode. You can tell Flash Player to use the traditional code page of the operating system that is running the player. To do this, add the following code as the first line of code in the first frame of the Flash application that is loading the data:

```
system.useCodepage = true;
```

Set the `system.useCodepage` property only once in a document; do not use it multiple times in a document to make the player interpret some external files as Unicode and some as other encoding, because doing so can yield unexpected results.

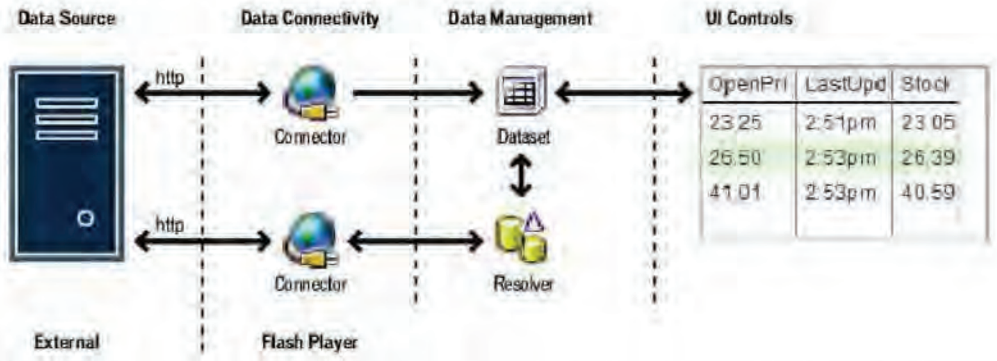
If you set the `system.useCodepage` property to `true`, keep in mind that the traditional code page of the operating system running the player must include the glyphs used in your external text file in order for the text to be displayed. For example, if you load an external text file that contains Chinese characters, those characters are not displayed on a system that uses the CP1252 code page, because that code page does not include Chinese characters. To ensure that users on all platforms can view external text files used in your Flash applications, you should encode all external text files as Unicode and leave the `system.useCodepage` property set to `false` by default. This causes Flash Player to interpret the text as Unicode. For more information, see `system.useCodepage` in ActionScript Dictionary Help.

CHAPTER 14
Data Integration (Flash Professional Only)

Macromedia Flash MX Professional 2004 provides a flexible component-based architecture and object model for connecting to external data sources, managing data, and binding data to user interface components. There are three main areas of data integration:

- Data connectivity: Connector layer of the architecture with components providing the ability to send and receive data from a variety of external data sources, such as web services and XML. See "Data connectivity (Flash Professional only)" on page 234.
- Data management: Data repository layer of the architecture with components providing intelligent supervision of common data operations, such as editing, sorting, filtering, aggregation, and translation of changes. See "Data management (Flash Professional only)" on page 242.
- Data binding: Data pipeline layer of the architecture providing a mechanism to share data between component properties. The pipeline integrates objects such as formatters and encoders to provide complete control over how the data is propagated between components. See "Data binding (Flash Professional only)" on page 256.

The following image depicts the flow of data within a Flash application and identifies the different elements that make up the Flash data architecture. Data binding is represented by the red arrows between the components.



Note: It is important to understand that Flash is a client-side technology. Flash data integration provides all the necessary tools and features to access your data, manipulate it, and send it back to a server in many different formats. Building and exposing business logic on the server is the job of a server developer and is best implemented using products that are specifically designed for that task (such as Cold Fusion, J2EE Application Servers, and ASP.NET).

For a tutorial that introduces Flash data architecture, see www.macromedia.com/go/data_integration.

Data connectivity (Flash Professional only)

Data connectivity in Flash MX Professional 2004 provides the ability to connect to external data sources to retrieve and send data. This functionality is provided within the Flash authoring environment through the connector and resolver components.

Note: External data refers to any data that is accessible through HTTP.

About data connectivity and security in Flash Player (Flash Professional only)

Access to external data through any connector component is subject to the sandbox security feature in Flash Player. This feature restricts a Flash document from accessing data from any domain other than the one in which it originated (this includes public web services). If you test an application from the authoring tool or display an application within a projector, you can access external data from any domain. However, if you publish a movie to www.abc.com, it can retrieve data only from www.abc.com.

Note: It is possible to access external data from another domain if it contains a policy file that states it can accept data requests that originate from your domain. See "Working with External Data" in ActionScript Reference Guide Help.

Connector components (Flash Professional only)

You use connector components to communicate with an external data source. Connector components contain specialized funtionality for working with a specific data source. Flash MX Professional 2004 includes the WebServiceConnector component, which enables you to connect to a web service, and the XMLConnector component, which enables you to connect to any external data source that returns XML through HTTP (such as JSP, ASP, Servlet, or ColdFusion). Connector components are used to obtain data from the server as well as send update packets back to the server. A typical screen in an application might contain several connector components for retrieving or updating data, or both.

The XMLConnector and WebServiceConnector components have no visual appearance in the runtime application. Advanced developers can build additional connector components as needed. For more information, see "Creating Components" in Using Components Help.

The WebServiceConnector component (Flash Professional only)

The WebServiceConnector component enables you to access remote methods exposed by a server using the industry-standard SOAP (Simple Object Access Protocol) protocol. A web service may accept parameters and return a result. Using the Flash MX Professional 2004 authoring tool and the WebServiceConnector component, you can introspect, access, and bind data between a remote web service and your Flash application. A single instance of a WebServiceConnector component can be used to make multiple calls to the same operation. To call more than one operation, though, you need to use a different instance of a WebServiceConnector component for each operation.

A web service's methods (sometimes referred to as operations) are defined by a Web Service Description Language (WSDL) file. The WSDL file specifies a list of operations, parameters, and results (referred to as a schema) that are exposed by the web service.

WSDL files are accessible using a URL. In Flash MX Professional 2004, you can view the schema of any web service by entering the URL for its WSDL file using the Web Services panel. Once you identify a WSDL file, the web service is available to any application you create.

Note: Keep in mind that access to a web service (as with any external data) is subject to Flash Player security features. See "About data connectivity and security in Flash Player (Flash Professional only)" on page 234.

For more information on the WebServiceConnector component, including its properties, methods, and events, see "WebServiceConnector component (Flash Professional only)" in Using Components Help.

WebServiceConnector component parameters

The following are the WebServiceConnector component parameters:

operation A string that is the name of a method of the web service. In the authoring UI, this appears as a pop-up menu, which is updated any time you change the WSDLURL.

multipleSimultaneousAllowed A Boolean value; when this parameter is set to `true`, it allows a `trigger()` operation to initiate when another `trigger()` operation is already in progress. Multiple simultaneous `trigger()` operations might not be completed in the same order they were called. Also, Flash Player may place limits on the number of simultaneous network operations. This limit varies by version and platform. When the parameter is set to `false`, a `trigger()` operation cannot initiate if another one is in progress.

suppressInvalidCalls A Boolean value; when this parameter is set to `true`, it suppresses the `trigger()` operation if the data parameters are invalid. When it is set to `false`, the `trigger()` operation executes and uses invalid data if necessary.

WSDLURL A string that is the URL of a web service WSDL file.

Common workflow for the WebServiceConnector component

The typical workflow for the WebServiceConnector component is as follows.

To use a WebServiceConnector component:

1 Use the Web Services panel to enter the URL for a web service WSDL file.

2 Add a call to a method of the web service by selecting the method, right-clicking (Windows) or Control-clicking (Macintosh), and selecting Add Method Call from the context menu. This creates a WebServiceConnector component instance in your application. The schema for the component can then be found on the Schema tab of the Component Inspector panel. You are free to edit this schema as needed—for example, to provide additional formatting or validation settings.

   Note: The schema for the `params` and `results` component properties is updated each time the author changes the `WSDLURL` or `operation` parameter. This overwrites any settings that you have edited.

3 Use the Binding tab in the Component Inspector panel to bind the web service parameters and results that are now defined in your schema to components within your application.

4 Add a trigger to initiate the data binding operation: use the Trigger Data Source behavior attached to a button, or add ActionScript.

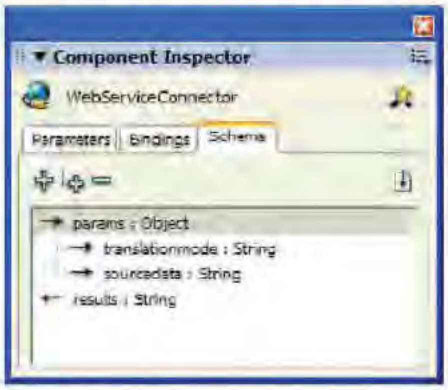
Schemas for web services

The schema for a web service is defined through its WSDL file. Flash MX Professional 2004 has built-in support for introspecting a WSDL and generating the schema.

The following example demonstrates how to view the schema for a web service.

Note: This example requires an active Internet connection, because it uses a public web service. If you use a web service in your application, the web service must be located in the same domain as the SWF file for your application in order for the application to work in a web browser. See "Applications and web services" on page 236.

1 Drag a WebServiceConnector component to the Stage.
2 Select the WSDLURL parameter, and type the following URL:
`http://www.xmethods.net/sd/2001/BabelFishService.wsdl`
3 Select Operation and select the BabelFish method.
4 Click the Schema tab. You will now see the following auto-generated schema:



This is a schematic representation of the service that you are calling. The parameters as well as result structure are defined within the schema. This schema states that the BabelFish Service expects two string parameters when it is called and will return a string as the result of the call.

The items identified within the schema can now be bound to a variety of UI controls to provide values for the parameters or to display the results of the web service.

Applications and web services

Many developers are interested in using an industry standard such as SOAP web services as the data-exchange mechanism between their client and server. One reason this approach is gaining favor is the increasing number of popular servers that support exposure of logic using SOAP.

However, there may be cases where you want the client software to consume web services that are published by third parties or hosted on servers that fall outside the Flash Player sandbox. There are several ways to make this work, while still preserving the end-user security and privacy that the Flash Player sandbox provides. These approaches involve the creation of an intermediary object that resides on the server to act as a bridge between your client and the public services you want to consume. This approach offers several advantages:

- Public web services can be aggregated. With this approach you can provide fail-over safety and load balancing when a request is made for data.
- You keep control over the flow of data in your application. If the web service goes away or the URL is down, you can decide how to respond.
- Data can be optimized. Multiple requests can be cached.
- You can have custom error handling. You can determine what errors to send back to the client.
- Data can be manipulated, converted, or combined. You can pull data from several sources and return one data packet with the combined information.

Many of the SOAP-based applications that you build will consume private web services hosted on your server. Once you have determined the best way to implement and expose your own web services, it is easy to make public web services available to your client application. When you are in control of the server, you can offer a complete solution. The server is the ideal place for business logic that can determine the best way to respond to requests for data and the results that should be sent back to the client. This is also the most secure way to build an application. The server can provide additional processing to make sure that users have access only to certain services as well as protect the client from making calls to malicious services that can return bad data. See "About data connectivity and security in Flash Player (Flash Professional only)" on page 234.

Lazy decoding

When the WebServiceConnector component receives multiple records of data from a web service, it translates them into an ActionScript array so that they are accessible within your application. Translating multiple records of data from XML/SOAP into ActionScript native data can be a very time-consuming process (large arrays can take seconds, or tens of seconds). The WebServiceConnector component supports a feature called lazy decoding, which defers this translation. Therefore, result values that are arrays are not immediately translated from XML to ActionScript. Instead, the result value passed to the user is a special object that acts similarly to an array and translates the XML data only when it is actually requested. You request the data by using the command `myArray[myIndex]`. The net effect of this feature is to improve the perceived performance of web services by spreading the workload over a longer period of time.

Note: The result value can be accessed only by using the `myArray[myIndex]` syntax. It cannot be accessed using a for..in loop such as `for (var i in myArray)`.

The XMLConnector component (Flash Professional only)

The XMLConnector component provides your application with access to any external data source that returns or receives XML through HTTP. The easiest way to connect with an external XML data source and use the parameters and results of that data source for your application is to specify a *schema*, the structure of the XML document that identifies the data elements in the document to which you can bind. You can manually create the schema through the Component Inspector panel, or you can use the authoring environment to create one automatically.

For more information on the XMLConnector component, including its properties, methods, and events, see "XMLConnector component (Flash Professional only)" in Using Components Help.

Note: The authoring environment accepts a copy of the external XML document you are connecting to as a model for the schema. If you are familiar with XML scripting, you can create a sample XML file that can be used to generate a schema.

XMLConnector component parameters

The following are the parameters for the XMLConnector component:

direction A string that defines what type of operation to perform via HTTP when `trigger()` is called: `send`, `sendAndLoad`, or `load` correspond to `receive`, `receive/send`, and `send`, respectively.

ignoreWhite A Boolean value; the default setting is `false`. When this parameter is set to `true`, the text nodes that contain only white space are discarded during the parsing process. Text nodes with leading or trailing white space are unaffected.

multipleSimultaneousAllowed A Boolean value; when this parameter is set to `true`, it allows a `trigger()` operation to initiate when another `trigger()` operation is already in progress. Multiple simultaneous `trigger()` operations may not be completed in the same order they were called. Also, Flash Player may place limits on the number of simultaneous network operations. This limit varies by version and platform. When the parameter is set to `false`, a `trigger()` operation cannot initiate if another one is in progress.

suppressInvalidCall A Boolean value; when this parameter is set to `true`, it suppresses the `trigger()` operation if the data parameters are invalid. When it is set to `false`, the `trigger()` operation executes and uses invalid data if necessary.

URL A string that points to an external XML data source.

Common workflow for the XMLConnector component

The typical workflow for the XMLConnector component is as follows.

To use an XMLConnector component:

1 Add an instance of the XMLConnector component to your application and give it an instance name.

2 Use the Parameters tab of the Component Inspector panel to enter the URL for the external XML data source that you want to access.

3 Use the Schema tab of the Component Inspector panel to specify a schema for the XML document.

Note: You can use the Import Sample Schema button to automate this process.

4 Use the Bindings tab of the Component Inspector panel to bind data elements (`params` and `results`) from the XML document to properties of the visual components in your application. For example, you can connect to an XML document that provides weather data and bind the Location and Temperature data elements to label components in your application. The name and temperature of a specified city appears in the application at runtime.

5 Add a trigger to initiate the data binding operation: use the Trigger Data Source behavior attached to a button, or add ActionScript.

Importing schemas for XML data sources

The key to using XML documents and data binding is to import a schema (the data structure that appears in the Schema panel) that you can then bind to.

To import a sample schema:

1 Drag an XMLConnector component to the Stage.

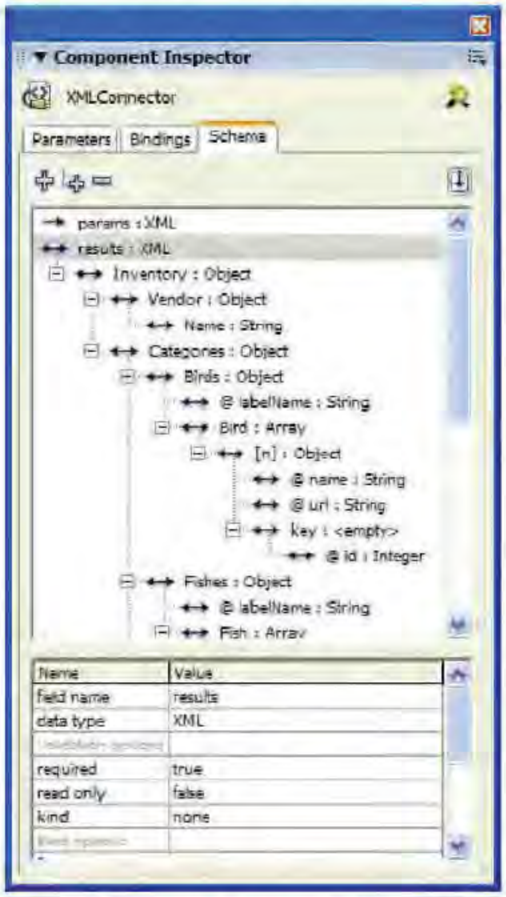
2 Click the schema tab of the Component Inspector panel and select Results.

3 Click the Import Sample Schema button in the upper right corner of the Schema tab to import a schema.

4 Select the file that you want to use as an example, and click Open.

The schema appears in the Schema tab

For example, the schema for a file named Animals.xml looks like the following:



```
params : XML
results : XML
  Inventory : Object
    Vendor : Object
      Name : String
    Categories : Object
      Birds : Object
        @ labelName : String
        Bird : Array
          [n] : Object
            @ name : String
            @ url : String
            key : <empty>
              @ id : Integer
      Fishes : Object
        @ labelName : String
        Fish : Array
```

Name	Value
field name	results
data type	XML
[illegible]	
required	true
read only	false
kind	none
[illegible]	

This is a schematic representation of the structure of the XML file. It says that the `results` property of the XMLConnector component is an XML object. The root element of that object is called Inventory, which contains the elements Vendor, categories, and so on. The Vendor element contains a single element called Name, which is a string. The categories field contains an element called Birds, which contains the attribute `labelname`. The Birds element also contains an array of objects called Bird. Each of these objects has two attributes: `name` and `url`. It also contains a single element named key, which contains the attribute `id`.

The String and Integer fields can be bound to a variety of UI components. The Array field Bird can be bound to List, DataGrid, or ComboBox components, or you can directly bind UI components to fields within the array. A typical workflow for an application that works with data would include binding an array from the XMLConnector component to the DataSet component's `dataProvider` property. In this scenario, the data set could be used to manage the data. The fields within the data set could then be mapped to any of the UI components using data binding.

Virtual XML schemas

If the structure of the XML data provided to your application is not of the desired form, a virtual schema allows you to change how the underlying data structure is interpreted when bindings are executed. The new structure is derived using XPath statements. See "Supported XPath expressions" on page 241.

For example, the schema for animals.xml file described previously (see Importing schemas for XML data sources) defines an array of objects called Bird. Each object contains two fields (`name` and `url`). They also contain a subelement with one field called `id`. If you bind the Bird array to a DataSet component (using the `dataProvider` property) with three fields—`name`, `url`, and `id`—each item that is returned from the array is constructed with this algorithm:

For each item in the XML:

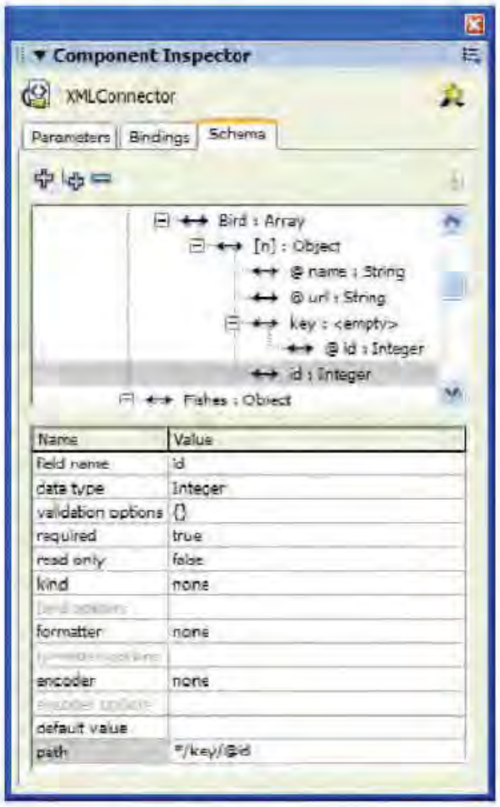
1 Create an empty item.

2 Loop through the defined schema properties, extracting the values for each property from the XML data, and assign these values to the created item. The Name and URL fields will have values.

3 Provide this item to the DataSet component.

The ID field does not exist on the item, and the DataSet component has a blank entry for each item assigned.

The solution is to create a new schema field under the object within the Bird array. The new schema field is named `id`. Every schema field has a property called `path` that accepts an XPath statement that points to the data in your XML file. In this case the XPath statement would be key/@id. When you get to step 2 in this procedure, data binding finds an `id` field for the object. It looks at the `path` property and uses the XPath statement to get the correct data from the XML file. The correct data is then passed to the DataSet component. For information on manually creating a schema, see "Working with schemas in the Schema tab (Flash Professional only)" on page 258.



Name	Value
field name	id
data type	Integer
validation options	{}
required	true
read only	false
kind	none
kind options	
formatter	none
formatter options	
encoder	none
encoder options	
default value	
path	*/key/@id

Supported XPath expressions

The following XPath expressions are supported:

- Absolute paths:

  `/A/B/C`
- Relative paths:

  `A/B/C`
- Node selection using node name or wildcard:

  `/A/B/C` (node selection by name)

  `/A/B/*` (node selection of all child nodes of /A/B by wildcard)

  `/*/*/C` (node selection of all C nodes that have exactly two ancestors)

- Predicate syntax to further specify nodes to be selected:

  `/B[C]` (child node syntax; selects all B nodes that have a C node as a child)

  `/B[@id]` (attribute existence syntax; selects all B nodes that have an attribute named `id`)

  `/B[@id="A1"]` (attribute value syntax; selects all B nodes that have an `id` attribute whose value is A1)
- Support for predicate comparison operators:

  =
- Support for Boolean `AND` and `OR` values in predicates:

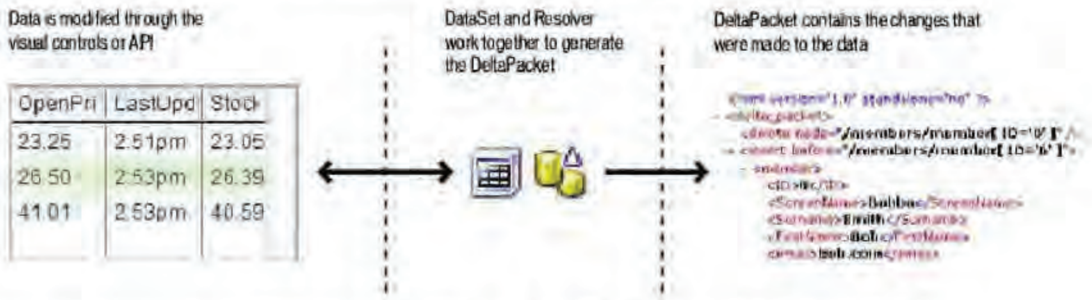
  `/B[@id=1 AND @customer="macromedia"]`

Note: The following operators are not supported: "<", ">", "//".

Data management (Flash Professional only)

Flash MX Professional 2004 provides developers with advanced functionality for managing data within a Flash application. The term *managed data* refers to the ability to perform advanced operations on a local cache of data, including multiple sorts, filters, finds, and offline caching. In addition, the changes that are made to your data through the UI components can be tracked and used to generate an optimized set of instructions (DeltaPacket) that can be manipulated into a specific format for consumption by a variety of external data sources.

Note: Method calls can be tracked, in addition to changes made to the data.


Data is modified through the visual controls or API
DataSet and Resolver work together to generate the DeltaPacket
DeltaPacket contains the changes that were made to the data

OpenPri	LastUpd	Stock
23.25	2:51pm	23.05
26.50	2:53pm	26.39
41.01	2:53pm	40.59

```
  <?xml version="1.0" standalone="no" ?>
- <delta_packet>
    <delete node="/members/member[ ID='0' ]" />
  - <insert before="/members/member[ ID='6' ]">
    - <member>
        <ID>[illegible]</ID>
        <ScreenName>Bobbo</ScreenName>
        <Surname>Smith</Surname>
        <FirstName>Bob</FirstName>
        <email>bob.com</email>
```

Managed versus unmanaged data (Flash Professional only)

There are two basic scenarios for working with data in the Flash authoring environment:

Unmanaged In this scenario, you bind the results of your connector component directly to UI components within your Flash document.

Managed You bind the results of your connector component to the DataSet component and bind the fields of the DataSet component to UI components within your Flash document. In addition, you can bind the DataSet component to a resolver component to format the changes to your data before they are sent to an external data source.

A managed data solution requires more setup but gives you greater control over your data. In general, you should use a managed data approach for the following scenarios:

- You plan to send updates back to an external data source using the built-in features of the DataSet and resolver components (such as automated tracking of changes to your data that can be converted into multiple formats).
- You need to apply multifield sorts, filters, or ranges to your data.
- You are building an application that provides the ability to work offline (changes to the data are cached offline and can be applied at a later time).
- You want to receive changes from the server and apply them to your local cache of data.
- You want to create your own transfer object implementation to complement a business class on the server.

The DataSet component (Flash Professional only)

The DataSet component is a specialized engine that manages a collection of items (transfer objects), each one representing a record of data from an external data source. Items within the collection can be iterated using filters, sorting, ranges, and random access. Using the DataSet component in conjunction with a connector and resolver component provides you with a complete solution for accessing, managing, and updating data between a Flash application and an external data source.

The DataSet component works only with Flash Player 7.

DataSet component parameters

The following are parameters that you can set for the DataSet component:

itemClassName A string that is the name of the transfer object class that is instantiated each time a new item is created within the DataSet component. You must make a fully qualified reference to this class somewhere within your code to make sure that it gets compiled into your application (such as `private var myItem:my.package.myItem;`).

The DataSet component uses transfer objects to represent the data that you retrieve from an external data source. If you leave this parameter blank, the data set creates an anonymous transfer object for you. If you give this parameter a value, the data set instantiates your transfer object whenever new data is added.

filtered A Boolean value that defaults to `false`. If this parameter is set to `true`, a filter is applied to the DataSet component so that it contains only the objects that match the filter criteria.

logChanges A Boolean value that defaults to `true`. If this parameter is set to `true`, the data set logs all changes to data or method calls.

readOnly A Boolean value that defaults to `false`. If this parameter is set to `true`, the data set cannot be modified.

Common workflow for the DataSet component

The typical workflow for the DataSet component is as follows.

To use a DataSet component:

1 Add an instance of the DataSet component to your application and give it an instance name.

2 Select the Schema tab for the DataSet component and create component properties to represent the persistent fields of the data set.

3 Load the DataSet component with data from an external data source. For more information, see "About loading the DataSet component" on page 244.

4 Use the Bindings tab of the Component Inspector panel to bind the data set fields to UI components within your application.

The UI controls are notified as a new record (transfer object) is selected within the DataSet component, and they are updated accordingly. In addition, the Dataset component is notified when data is modified within a UI control, and the change is tracked using the DeltaPacket.

5 Call the methods of the DataSet component within your application to manage your data.

Note: In addition to these steps, you can also bind the DataSet component to a connector and a resolver component to provide a complete solution for accessing, managing, and updating data from an external data source.

For more information on the DataSet component, see "DataSet component (Flash Professional only)" in Using Components Help.

About loading the DataSet component

It is important to remember that the DataSet component is a collection of transfer objects. This differs from previous implementations of the component, when it was simply an in-memory cache of data (array of record objects). Transfer objects expose business data from an external data source through public properties or accessor methods. The DataSet component allows enterprise developers to work with sophisticated client-side objects that mirror their server-side counterparts, or in its simplest form, a collection of anonymous objects with public properties representing the fields within a record of data.

There are two data structures that can be used to load data into the DataSet component:

- An array of objects
- An object that implements the DataProvider interface

An array of objects can be bound or assigned in code to the `DataSet.items` property. A class that implements a DataProvider interface can be bound or assigned in code to the `DataSet.dataProvider` property.

The following are examples of loading objects into the DataSet component:

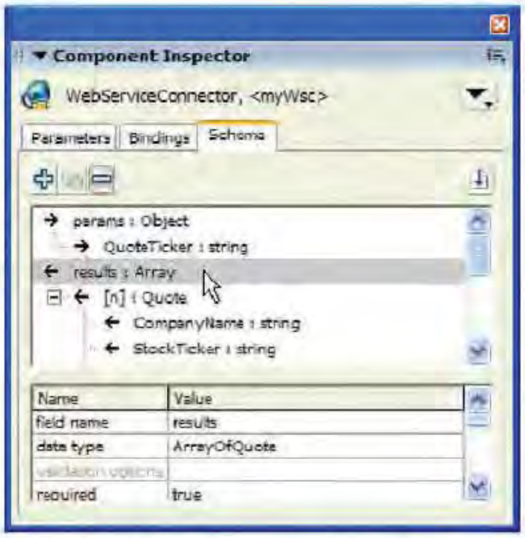
Anonymous objects This example assigns 100 anonymous objects to the DataSet component. Each object represents a record of data.

```
function loadData() {
   var recData = new Array();
   for( var i:Number=0; i<100; i++ ) {
      recData[i]= {id:i, name:String("name"+i), price:i*.5};
   }
   myDataSet.items = recData;
}
```

Remoting RecordSet This example assumes that you've made a remoting call that returns a RecordSet. The RecordSet object implements the DataProvider interface.

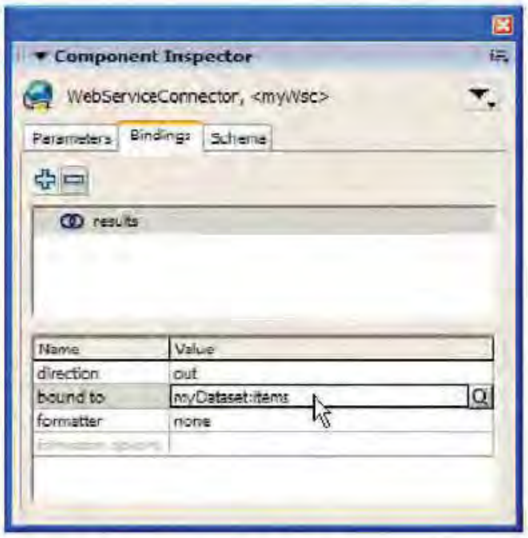
```
function getSQLData_Result(result) {
  myDataset.dataProvider = result;
}
```

Array of objects returned from a web service Using data binding, you can assign an array of objects returned from the web service to the DataSet component's `items` property.



params : Object
QuoteTicker : string
results : Array
[n] : Quote
CompanyName : string
StockTicker : string

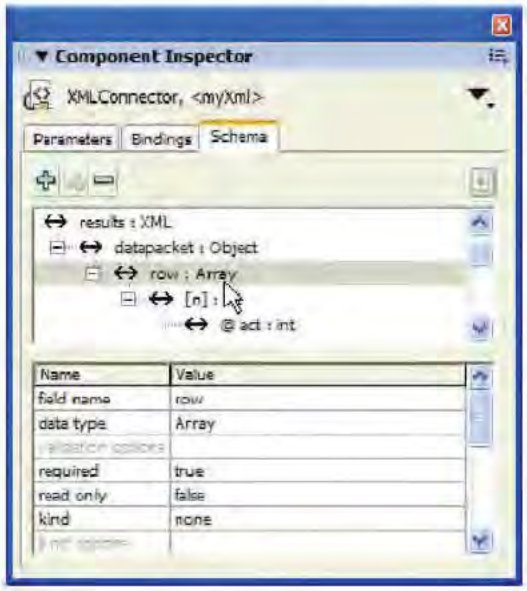
Name	Value
field name	results
data type	ArrayOfQuote
validation options	
required	true



results

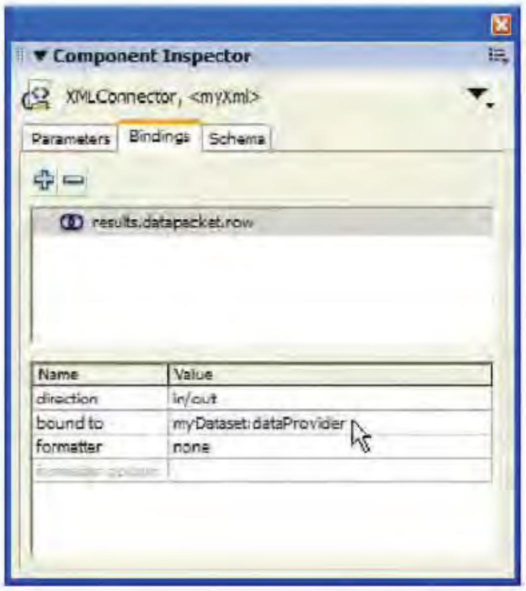
Name	Value
direction	out
bound to	myDataset:items
formatter	none
formatter options	

Array of objects returned from an XMLConnector component This example assumes that you have read in a schema for an XML file that contains an array of objects. In this scenario, the actual data being returned from the XMLConnector component is an array of XML nodes, which is not supported by the DataSet component. However, data binding (the transport mechanism that is used to copy the data from the XMLConnector component to the DataSet component) implements the DataProvider interface. Therefore, you can assign the array of XML nodes to the `DataSet.dataProvider` property, and the data binding feature does the rest.



results : XML
datapacket : Object
row : Array
[n] :
@ act : int

Name	Value
field name	row
data type	Array
validation options	
required	true
read only	false
kind	none
kind options	



results.datapacket.row

Name	Value
direction	in/out
bound to	myDataset:dataProvider
formatter	none
formatter options	

Transfer objects

When you load data into the DataSet component, the date get translated into a collection of transfer objects. In the simplest scenario, the DataSet component creates and loads the data into anonymous objects. Each anonymous object implements the TransferObject interface, which is all that is required for the DataSet component to manage the objects. The DataSet component tracks changes made to the data and any method calls that are made on the objects. If methods are called on an anonymous object, nothing happens, because the methods don't exist. However, the DataSet component tracks them in the DeltaPacket, which guarantees that they will be sent to the external data source, where they can be called if appropriate.

In an enterprise solution you could create a client-side ActionScript transfer object that mirrors a server-side transfer object. This client object could implement additional methods for manipulating the data or applying client-side constraints. Developers can use the `itemClassName` parameter of the DataSet component to identify the class name of the client-side transfer object that should be created. In this scenario, the DataSet component generates multiple instances of the specified class and initializes it with the loaded data. When `addItem()` is called on the DataSet component, the `itemClassName` is used to create an empty instance of the client-side transfer object.

If you take the enterprise solution one step further, you could implement a client-side transfer object that makes use of web services or Flash Remoting. In this scenario, the object would make direct calls on the server in addition to possibly storing the calls in the DeltaPacket.

Note: You can create your own custom transfer object for use by the DataSet component by creating a class that implements the TransferObject interface. For information on the TransferObject interface, see the "Components Dictionary" in Using Components Help.

Accessing the data

Once the data is loaded into the DataSet component, it needs to be accessed. Accessing the data at runtime is simple. Transfer objects expose data through properties that can be referenced in code. The DataSet component has a concept of a cursor that points to the currently selected transfer object. The following code loads a DataSet component with customer information and then displays each customer's name in the trace window:

```
var recData = [{id:0, firstName:"Frank", lastName:"Jones", age:27,
  usCitizen:true},
             {id:1, firstName:"Susan", lastName:"Meth", age:55,
  usCitizen:true},
             {id:2, firstName:"Pablo", lastName:"Picasso", age:108,
  usCitizen:false}];
myDataSet.items = recData;
myDataSet.first();
while ( myDataSet.hasNext() ) {
  //access the data through the Dataset attributes
  trace(myDataSet.firstName + " " + myDataSet.lastName);
  myDataSet.next();
}
```

To set up the binding to this data at design time, you create persistent fields for the DataSet component that represent the properties of the transfer object, as in the following example.

To access data at design time:

1 Drag a DataSet component onto the Stage. Name it **myDataSet**.

2 Select a layer in the Timeline and press F9 to open the Actions panel. Type the following code:

```
var recData = [{id:0, firstName:"Frank", lastName:"Jones", age:27,
  usCitizen:true},
                {id:1, firstName:"Susan", lastName:"Meth", age:55,
  usCitizen:true},
                {id:2, firstName:"Pablo", lastName:"Picasso", age:108,
  usCitizen:false}];
myDataSet.items = recData;
```

3 With the DataSet component selected, click the Schema tab of the Component Inspector panel and click the Add a Component Property (+) button.

4 Set the value for Field Name to **firstName** and leave the Data Type as String.

5 Create two more component properties (Field Name = lastName, Data Type = String) and (Field Name = usCitizen, Data Type = Boolean).

6 Drag a DataGrid component onto the Stage and name it **myGrid**.

7 Select the DataGrid component, and click the Bindings tab of the Component Inspector panel.

8 Click the Add Binding (+) button to add a new binding. Select "dataProvider."

9 Click Bound To and select the `dataProvider` property of the DataSet component.

10 Click Direction and select In.

11 Run the application.

The data contained within the transfer objects within the data set is now displayed in the data grid.

Creating fields for a DataSet component at design time is the easiest way to expose the properties of a transfer object to data binding. Once the fields are defined, you can visually bind UI controls to the data at design time. Also, there are many additional properties (schema item settings) that can be set at design time for a DataSet field that effect the way data is encoded, formatted, and validated at runtime. See "Working with schemas in the Schema tab (Flash Professional only)" on page 258.

The ability to make use of dynamic component properties that are added to the Schema tab at design time is a special feature of the DataSet component. The DataSet component uses the field name of these properties to map them to the properties of the transfer object. The settings that are applied to these properties at design time are then used by the data set at runtime.

If you do not create persistent fields for the DataSet component and you bind it to a WebServiceConnector component or an XMLConnector component that defines a schema, the DataSet component tries to create the correct fields based on the connector component's schema.

Note: Persistent fields that are defined for a DataSet component take precedence over the schema for a connector component.

Resolver components (Flash Professional only)

You use the resolver components in combination with the DataSet component (part of the data management functionality in the Flash data architecture). The resolver components enable you to convert changes made to the data within your application into a format that is appropriate for the external data source that you are updating. Flash MX Professional 2004 includes the XUpdateResolver and RDBMSResolver components. The XUpdateResolver component targets XML data sources, and the RDBMSResolver component targets relational databases. These components have no visual appearance at runtime. Developers can also build additional resolver components as needed.

If you use a DataSet component in your application, it generates an optimized set of instructions (DeltaPacket) describing the changes made to the data at runtime. This set of instructions is converted to the appropriate format (update packet) by the resolver components. When an update is sent to the server, it is possible that the server will send a response (result packet) containing additional updates or errors as a result of the update operation. The resolver components can convert this information back into a DeltaPacket that can then be applied to the data set to keep it in sync with the external data source. Resolver components enable you to keep your application and an external data source in sync without writing additional ActionScript code.

The XUpdateResolver component (Flash Professional only)

The XUpdateResolver component converts changes made to the data in your application into XUpdate statements that can be processed by an external data source. XUpdate is a standard for describing changes that are made to an XML document and is supported by a variety of XML databases, such as Xindice and XHive.

Note: You can also use the XUpdateResolver component to send data updates to any external data source that can parse the XUpdate language–for example, an ASP page, a Java servlet, or a ColdFusion component. For more information, see the XUpdate specification at www.xmldb.org/xupdate/.

The XUpdateResolver component communicates with the DataSet component using the DataSetDeltaToXUpdateDelta encoder. This encoder is responsible for creating XPath statements that uniquely identify nodes within an XML file based on the information contained within the DataSet component's DeltaPacket. This information is used by the XUpdateResolver component to generate XUpdate statements. For more information on the DataSetDeltaToXUpdateDelta encoder, see "Schema encoders (Flash Professional only)" on page 267.

The XUpdateResolver component works only with Flash Player 7.

XUpdateResolver component parameter

The XUpdateResolver component has one parameter, the `includeDeltaPacketInfo` parameter (Boolean type). When this parameter is set to `true`, the update packet includes additional information from the data set that can be used by an external data source to generate results that can be sent back to your application. This information includes a unique transaction and operation ID that is used internally by the data set.

Note: The additional information that is included in the update packet invalidates the XUpdate. This is expected behavior. You would choose to add this information only if you were going to store it within a server object and use it to generate a result packet. In this scenario, your server object would pull the information out of the update packet for it s own needs and then pass on the (now valid) XUpdate to the database.

Example (with `false` setting):

```
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/
  xupdate">
    <xupdate:remove select="/datapacket/row[@id='100']"/>
</xupdate:modifications>
```

Example (with `true` setting):

```
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/
  xupdate"
      transId="46386292065:Wed Jun 25 15:52:34 GMT 0700 2003">
    <xupdate:remove select="/datapacket/row[@id='100']" opId="0123456789"/>
</xupdate:modifications>
```

Common workflow for the XUpdateResolver component

The typical workflow for the XUpdateResolver component is as follows.

To use an XUpdateResolver component:

1 Add two instances of the XMLConnector component and one instance each of the DataSet component and XUpdateResolver component to your application and give them instance names.

2 Select the first XMLConnector component, and use the Parameters tab of the Component Inspector panel to enter the URL for the external XML data source that you want to access.

3 With the XMLConnector component still selected, click the Schema tab of the Component Inspector panel and import a sample XML file to generate your schema.

Note: You may need to create a virtual schema for your XML file if you want to access a subelement of the array that you are binding to the data set. For more information, see "Virtual XML schemas" on page 240.

4 Use the Bindings tab of the Component Inspector panel to bind an array within the XMLConnector component to the `dataProvider` property of the DataSet component.

5 Select the DataSet component and use the Schema tab of the Component Inspector panel to create the DataSet fields that will be bound to the fields of the object within the array.

6 Use the Bindings tab of the Component Inspector panel to bind data elements (DataSet fields) to the visual components in your application.

7 Select the Schema tab of the XUpdateResolver component. With the `deltaPacket` component property selected, use the Schema Attributes pane to set the `encoder` property to the DataSetDeltaToXUpdateDelta encoder.

8 Select Encoder Options and enter the `rowNodeKey` value that uniquely identifies the row node within the XML file.

Note: The `rowNodeKey` value combines an XPath statement with a field parameter to define how unique XPath statements should be generated for the update data contained within the DeltaPacket. See information on the DataSetDeltaToXUpdateDelta encoder in "Schema encoders (Flash Professional only)" on page 267.

9 Click the Bindings tab and create a binding between the XUpdateResolver component's `deltaPacket` property and the DataSet component's `deltaPacket` property.

10 Create another binding from the `xupdatePacket` property to the second XMLConnector component to send the data back to the external data source.

Note: The `xupdatePacket` property contains the formatted DeltaPacket (XUpdate statements) that will be sent to the server.

11 Add a trigger to initiate the data binding operation: use the Trigger Data Source behavior attached to a button, or add ActionScript.

Note: In addition to these steps, you an also create bindings to apply the result packet sent back from the server to the data set via the XUpdateResolver component.

For more information, see "XUpdateResolver component (Flash Professional only)" in Using Components Help.

Updates sent to an external data source

Update packets sent to the server are in XUpdate format. The following is a sample XUpdate packet:

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/
  xupdate">
   <xupdate:insert-after select="/addresses/address[1]" >
      <xupdate:element name="address">
         <xupdate:attribute name="id">2</xupdate:attribute>
         <fullname>Lars Martin</fullname>
         <born day='2' month='12' year='1974'/>
         <town>Leizig</town>
         <country>Germany</country>
      </xupdate:element>
   </xupdate:insert-after>
</xupdate:modifications>
```

Receiving results from an external data source

Once the server has finished with the update packet, either successfully or unsuccessfully, it should send back a result packet containing errors or additional XML updates resulting from the update operation. If there are no messages, the results packet should still be sent, but it will have no operation result nodes.

The following is a sample result packet for an update packet that has no errors and contains no XML updates:

```
<results_packet nullValue="{_NULL_}" transID="46386292065:Wed Jun 25 15:52:34
  GMT-0700 2003"/>
```

A sample results packet (with XML updates) follows:

```
<results_packet nullValue="{_NULL_}" transID="46386292065:Wed Jun 25 15:52:34
  GMT-0700 2003">
    <operation op="remove" id="11295627479" msg="The record could not be
  found"/>
    <operation op="update" id="02938027477">
       <attribute name="id" curValue="105" msg="Invalid field value" />
    </operation>
 </results_packet>
```

The results packet can contain an unlimited number of operation nodes. Operation nodes contain the results of operations from the update packet. Each operation node should have the following attributes/child nodes:

- `op`: An attribute describing the type of operation that was attempted. Must be insert, delete, or update.
- `id`: An attribute that holds the ID from the operation node that was sent out
- msg (optional): An attribute containing a message string that describes the problem that occurred when attempting the operation
- `field`: 0, 1, or more child nodes that give field-level specific information. Each field node, at a minimum, should have a `name` attribute, which contains the field name, and a `msg` attribute, which gives the field-level message. It may also optionally contain a `curValue` attribute, which holds the most up-to-date value for that field in that row on the server.

The RDBMSResolver component (Flash Professional only)

The RDBMSResolver component creates an XML packet that can be sent to an external data source (such as ASP/JSP page, servlet, and so on). The XML packet can easily be translated into SQL statements that can be used to update any standard SQL relational database

Note: You can use the RDBMSResolver component to send data updates to any external data source that can parse XML and generate SQL statements against a database –for example, an ASP page, a Java servlet, or a ColdFusion component.

The RDBMSResolver component works only with Flash Player 7.

RDBMSResolver component parameters

The following are the RDBMSResolver component parameters:

tableName The table name put in the XML for the DB table that should be updated. This should be the same as the value input for any `fieldInfo` items (or they should be blank) for any fields that should be updated.

updateMode This parameter has several values that determine the way key fields are identified when the XML update packet is being generated, as follows:

`umUsingAll`: This setting uses the old values of all of the fields within the FxDataset to identify the record to be updated. This is the safest approach to updating, because it lets you guarantee that another user has not modified the record since you retrieved it. However, this approach is more time-consuming and generates a larger update packet.

`umUsingModified`: This setting uses the old values of all of the fields that were modified within the FxDataset to identify the record to be updated. This method lets you guarantee that another user has not modified the same fields in the record since you retrieved it.

`umUsingKey`: This is the default value for this parameter. This setting uses the old value of the key fields within the FxDataset. This implies an optimistic concurrency model, which most database systems today employ. This method guarantees that you are modifying the same record that you retrieved from the database. Your changes overwrite any other user's changes to the same data.

nullValue The value put in a field's value to indicate a null value. This is customizable to prevent it from being confused with an empty string ("") or another valid value.

fieldInfo If your data source is a database table, then it should have one or more key fields that uniquely identify a record within a table. Additionally, there may be fields that have been calculated or joined from another table. Those fields must be identified so that key fields can be set within the XML update packet, and fields that should not be updated are left out of the XML update packet.

The RDBMSResolver component provides a `fieldInfo` parameter for this purpose. This parameter allows you to define an unlimited number of fields that require special handling. Each `fieldInfo` item contains three properties:

`fieldName`: Name of a field. This should map to a field in the data set.

`ownerName` (optional): This is used to identify fields that are not "owned" by the same table that is defined in the component `TableName` parameter. If this is filled in with the same value as that parameter or left blank, then the field is included normally in the XML update packet. If filled in differently, then this field is left out of the update packet.

`isKey`: A Boolean value that should be set to `true` for all key fields for the table that will be updated.

The following example identifies the key fields in the customer table. The customer table has a single key field, `"id"`, so you should create a `fieldInfo` item with the following values:

```
fieldName = "id", ownerName = leave this value blank, isKey = "true"
```

As a further example, suppose a field `"custType"` was added via a join in the query, and you don't want that field included in the update. You can create a field item with the following values:

fieldName = "custType", ownerName = "joinedField", isKey = "false"

Once the field items are defined, the RDBMSResolver component can use them to automatically generate the XML update packet to be sent to your external data source.

Note: The `FieldInfo` parameter makes use of a feature in Flash called the Collection Editor. When you select the `FieldInfo` parameter, the Collection Editor dialog box opens. From here you can add new `FieldInfo` items and set their `fieldName`, `ownerName`, and `isKey` properties from one location.

Common workflow for the RDBMSResolver component

The typical workflow for the RDBMSResolver component is as follows.

To use an RDBMSResolver component:

1 Add two instances of the WebServicesConnector component and one instance each of the DataSet and RDBMSResolver components to your application and give them instance names.

2 Select the first WebServicesConnector component and use the Parameters tab of the Component Inspector panel to enter the WSDLURL for a web service that exposes data from an external data source.

Note: The web service must return an array of records to be bound to the data set.

3 Use the Bindings tab of the Component Inspector panel to bind the WebServicesConnector component to the DataSet component.

4 Select the DataSet component, and use the Bindings tab of the Component Inspector panel to bind data elements (DataSet fields) to the visual components in your application.

5 Bind the DataSet component to the RDBMSResolver component.

Note: The update instructions are sent from the DataSet component to the RDBMSResolver component when the `applyUpdates` method of the DataSet component is called.

6 Select the RDBMSResolver component and bind it to the second WebServiceConnector component to send the data back to your external data source.

7 Add a trigger to initiate the data binding operation: use the Trigger Data Source behavior attached to a button, or add ActionScript.

Note: In addition to these steps, you an also create bindings to apply the result packet sent back from the server to the DataSet component via the RDBMSResolver component.

For more information, see "RDBMSResolver component (Flash Professional only)" in Using Components Help.

About sending updates to an external data source

The XML update packet contains three different types of nodes: `delete`, `insert`, and `update`. Each of these nodes represents a change to a row in a database table. Each node contains field nodes that locate a record to update and describe the modifications made, if any.

The updates sent from a resolver component are in the form of an XML update packet that is sent to an external data source via a connector component. The following is an example of an RDBMSResolver component's XML update packet generated with `updateMode` parameter set to `umUsingKey`:

```
<update_packet tableName="customers" nullValue="{_NULL_}"
  transID="46386292065:Wed Jun 25 15:52:34 GMT-0700 2003">
      <delete id="11295627477">
         <field name="id" type="numeric" oldValue="10" key="true"/>
      </delete>
      <insert id="12345678901">
         <field name="id" type="numeric" newValue="20" key="true"/>
         <field name="firstName" type="string" newValue="Davey" key="false"/>
         <field name="lastName" type="string" newValue="Jones" key="false"/>
      </insert>
      <update id="98765432101"> <field name="id" type="numeric" oldValue="30"
  key="true"/>
         <field name="firstName" type="string" oldValue="Peter"
  newValue="Mickey" key="false"/>
        <field name="lastName" type="string" oldValue="Tork" newValue="Dolenz"
  key="false"/>
      </update>
   </update_packet>
```

Elements in the XML update packet include the following:

- `transID`: An ID generated by the DeltaPacket that uniquely identifies this transaction. This information should accompany the results packet returned to this component.
- `delete`: This type of node contains information about a row that was deleted.
- `insert`: This type of node contains information about a row that was added.
- `update`: This type of node contains information about a row that was modified.
- `id`: A number that uniquely identifies the operation within the transaction. This information should accompany the results packet returned to this component.
- `newValue`: This attribute contains the new value for a field that was modified. It appears only when the field value has changed.
- `key`: This attribute is `true` if the field should be used to locate the row to update. This value is determined by the combination of the RDBMSResolver component's `updateMode` parameter, the `fieldInfo.isKey` setting, and the type of operation (insert, delete, update).

The following table describes how the key attributes value is determined. If a field is defined as a key field, using the RDBMSResolver component's `fieldInfo` parameter, it will always appear in the update packet with `key="true"`. Otherwise, the field's key attribute in the update packet will be set according to the following table:

Node type	umUsingKey	umUsingModified	umUsingAll
delete	`false`	`true`	`true`
insert	`false`	`true`	`false`
update	`false`	`true` if the field was modified, `false` otherwise	`true`

About receiving results from an external data source

Once the server has finished with the update packet, either successfully or unsuccessfully, it should send back a result packet containing errors or additional updates resulting from the update operation. If there are no messages, the results packet should still be sent, but it will have no operation result nodes.

The following is a sample RDBMSResolver component results packet (with both update results and change information nodes):

```
<results_packet nullValue="{_NULL_}" transID="46386292065:Wed Jun 25 15:52:34
  GMT-0700 2003">
      <operation op="delete" id="11295627479" msg="The record could not be
  found"/>
      <delete>
         <field name="id" oldValue="1000" key="true" />
      </delete>
      <insert>
         <field name="id" newValue="20"/>
         <field name="firstName" newValue="Davey"/>
         <field name="lastName" newValue="Jones"/>
      </insert>
      <operation op="update" id="02938027477" msg="Couldn't update employee.">
         <field name="id" curValue="105" msg="Invalid field value" />
      </operation>
      <update>
         <field name="id" oldValue="30" newValue="30" key="true" />
         <field name="firstName" oldValue="Peter" newValue="Mickey"/>
         <field name="lastName" oldValue="Tork" newValue="Dolenz"/>
      </update>
   </results_packet>
```

The results packet contains four different types of nodes:

- `Operation` nodes contain the result of operations from the update packet. Each operation node should have the following attributes/child nodes:

  `op`: An attribute describing the type of operation that was attempted. Must be insert, delete, or update.

  `id`: An attribute that holds the ID from the operation node that was sent out

  `msg` (optional): An attribute containing a message string that describes the problem that occurred when attempting the operation

  `field`: 0, 1, or more child nodes that give field-level specific information. Each field node, at a minimum, should have a `name` attribute that contains the field name, and a `msg` attribute that gives the field-level message. It may also optionally contain a `curValue` attribute that holds the most up-to-date value for that field in that row on the server.

- `Update` nodes contain information about records that have been modified since the client was last updated. Update nodes should have child nodes that list the fields that are necessary to uniquely identify the record that was deleted, and that describe fields that were modified. Each field node should have the following attributes:

  `name`: Holds the name of the field

  `oldValue`: Holds the old value of the field before it was modified. This attribute is required only when the key attribute is included and set to `true`.

  `newValue`: Holds the new value that the field should be given. This attribute should not be included if the field was not modified (i.e., the field has been included in the list only because it is a key field).

  `key`: Holds a Boolean `true` or `false` value that determines whether or not this field will be used as a key to locate the corresponding record on the client. This attribute should be included and set to `true` for all key fields. It is optional for all others.

- `Delete` nodes contain information about records that have been deleted since the client was last updated. Delete nodes should have child nodes that list the fields that are necessary to uniquely identify the record that was deleted. Each field node must have a `name` attribute, an `oldValue` attribute, and a `key` attribute whose value is set to `true`.

- `Insert` nodes contain information about records that have been added since the client was last updated. Insert nodes should have child nodes that describe the field values that were set when the record was added. Each field node must have a `name` attribute and a `newValue` attribute.

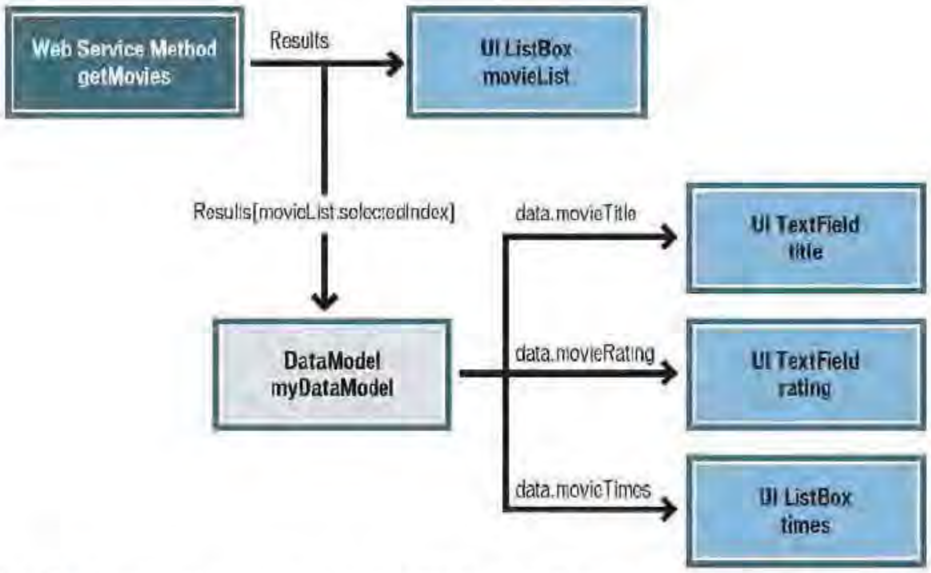
The DataHolder component (Flash Professional only)

The DataHolder component is a simplified version of the DataSet component whose only purpose is to hold data. It can be used as a connector between components. It communicates with other components via data binding. Initially the DataHolder component has a single bindable property, `data`. The user can add more properties, using the schema panel, if desired. The DataHolder component has no runtime visual appearance.

Every bindable property of the DataHolder component, including the built-in property `data` or any other property you add, works as follows: you can assign any type of data to a DataHolder property, either through data binding or using your own ActionScript code. Whenever you do, the DataHolder component emits an event whose name is the same as the property, and it executes any bindings if appropriate.

In most cases, you will not use this component to build an application. It is needed only when, for some reason, you cannot bind external data directly to another component and you do not want to use a DataSet component. Here are some situations in which you would use the DataHolder component:

- For a data value that is generated by ActionScript code but that you want to bind to some other component. For this case, you could have a DataHolder component that is databound as required; your ActionScript code can assign new values at any time to the component, and these values will be distributed.
- For a data value that can come (via data binding) from several sources (for example, several web services that all return the same kind of query result). In this case you could bind data from all the sources to the DataHolder component and then bind from the DataHolder component to the UI components that are displaying the data.
- For a data value that results from a complex, indexed data binding, as in the following schematic diagram. In this case, it is convenient to bind the data value to a DataHolder component and then use that for bindings to the UI.



Note: The DataHolder component is not meant to implement the same control over your data as the DataSet component. It does not manage or track data, or have the ability to update data. It is a repository for holding data and generating events when that data has changed.

Data binding (Flash Professional only)

Data binding is way of connecting components to each other. Components can be viewed as simple black boxes that have properties. A property is what lets you get data into and out of a component. A *binding* is a statement that says "When property X of component A changes, copy the new value to property Y of component B." You do data binding within the authoring tool using the Bindings tab of the Component Inspector panel. From here you can add, view, and remove bindings for a component.

Although data binding works with any component, its main purpose is to connect UI components to external data sources such as web services and XML documents. These external data sources are available as components with properties, which you can bind to other component properties. The Component Inspector panel is the main tool that is used within Flash MX Professional 2004 for data binding. It contains a Schema tab for defining the schema for a component and a Bindings tab for creating bindings between component properties.

Caution: If you copy and paste components, their bindings will be lost. You will need to reset the bindings manually. This is also the case if you select several components and convert them into a symbol.

The following example demonstrates how to create basic data binding by connecting one UI component to another.

To connect UI components to create data binding:

1 Add a NumericStepper component to your application and name it **A**.

2 Add another NumericStepper component and name it **B**.

3 With A selected, open the Component Inspector panel and click the Bindings tab.

4 Click the Add Binding (+) button to add a binding.

5 In the Add Binding dialog box, select Value, and click OK.

6 In the Name/Value section at the bottom of the Bindings tab, click the Bound To item under Name, and click the magnifying glass icon across from the Bound To item under Value.

7 In the Bound To dialog box, under Component Path select component B, and click OK.

8 Select Control > Test Movie. Click the Up and Down buttons on component A.

Each time you click the buttons on A, the `value` property of A is copied to the `value` property of B. Each time you click the buttons on B, the `value` property of B is copied to the `value` property of A.

9 Return to editing the application.

10 Create another NumericStepper component and name it **C**.

11 .Create a Text Input component called D.

12 Repeat steps 4-7 and bind the `value` property of C to the `text` property of D.

13 Select Control > Test Movie. Type a number in the text input field, and then press Tab.

Each time you enter a new value, the `text` property of D is copied to the `value` property of C. When you click the Up and Down buttons on C, the `value` property of C is copied to the `text` property of D.

Note: Data binding is supported only between components that exist in Frame 1 of the main Timeline, Frame 1 of a movie clip, or Frame 1 of a screen.

For a tutorial that introduces Flash data architecture, see www.macromedia.com/go/data_integration.

Working with schemas in the Schema tab (Flash Professional only)

The Schema tab allows you to view the schema for the selected component. The schema contains a list of the component's bindable properties, their data types, their internal structure, and various special attributes. This is the information that the data binding feature needs in order to handle your data correctly.

The Schema Tree pane, the top pane of the Schema tab, displays the bindable properties of the selected component and their internal structure, represented by one or more schema fields (component properties and schema fields are also referred to as schema items). The Schema Attributes pane, the bottom pane of the Schema tab, displays detailed information about the selected schema item.

Note: All components have properties, but by default, to reduce UI clutter, the Schema tab shows only those properties that commonly contain dynamic data. These properties are referred to as *bindable properties*. You can bind to any property by either adding it to the schema panel yourself or using ActionScript code. See "Working with bindings in the Bindings tab (Flash Professional only)" on page 272.

A component's schema describes the structure and type of data but is independent of how the data is actually stored. For example, the results from a WebServiceConnector component or an XMLConnector component could have identical schemas, even though the web service results are stored as ActionScript data structures (objects, arrays, strings, Boolean values, and numbers), and the XMLConnector component results are stored as XML objects. When you use data binding to access fields within a component's schema, you use the same procedure regardless of how the data are stored.

A component identifies which of its properties are bindable. These bindable properties appear in the Schema panel as top-level schema items (component properties). A component property can have its own internal structure (schema) that defines additional properties (schema fields) that can be bound to other component properties within your application. A good example of this is when you introspect a WSDL for a WebServiceConnector component. The WSDL definition describes the parameters and the results for a web service. The WebServiceConnector component contains two bindable properties (`params` and `results`). When the WebServiceConnector component introspects, the WSDL the authoring tool automatically creates the schema for the `params` and `results` properties so that it mirrors the schema defined within the WSDL.

There are several ways to create the schema for a component:

- For some components, you can import an XML sample file to define a property's schema.
- For some components, the schema is predefined within the component.
- For some components, the schema is derived as the result of introspecting an external schema definition such as the WSDL for a web service.
- For some components, you define the schema using the Schema toolbar within the Schema panel.

Importing a sample XML file to use as a schema

When you import a sample XML file to use as a schema, Flash MX Professional 2004 examines the sample document and creates a schema that represents the nesting structure of the document's XML elements and attributes, as well as the data type (number, Boolean value, or string) of the text values. Any element that occurs more than once appears in the schema as an array.

To import a sample XML file to define the schema for a component property:

1 Select the component property.

2 Do one of the following:

- Click the Import Sample Schema button in the upper right corner of the Schema tab.
- Click the options menu control in the upper right corner of the Component Inspector panel and select Import XML Schema from the menu.

3 In the Open File dialog box, select an XML file that is a representative sample of the data.

The schema appears in the Schema tab. You can now create a binding between your XML element and the component property within your application.

Note: Some XML documents may have a structure that Flash MX Professional 2004cannot represent—for example, elements that contain text and child elements mixed together.

Adding a schema field to a schema item

Use the following procedure to add a schema field to a schema item.

To add a schema field to a schema item:

1 In the Schema tab, select the schema item to which you want to add a field.

2 Click the Add a Field Under the Selected Field (+) button.

A new field is added as a subfield of the selected property.

3 In the Schema Attributes pane, enter a value for Field Name. Fill in the other attributes as appropriate.

There are three possible scenarios based on the type of schema item:

- Schema item of type Object, which may have subfields, @attributes, or both
- Schema item of type Array, which always has exactly one subfield called [n], which can be of any type (including Object, String, and so on)
- Schema item of other types (such as Boolean, String, Number), which don't have subfields but may have @attributes

Adding a component property to a schema

You can make any of your component properties bindable if you add them to the schema. The following procedure explains how to create an application that uses a CheckBox component to indicate whether a TextInput component is editable. However, in this example, the `editable` property of the TextInput component is not bindable, so its schema must be modified.

To add a component property to a schema:

1 Add an instance of a TextInput component and a CheckBox component to your application and give them instance names.

2 Select the TextInput component and click the Schema tab on the Component Inspector panel.

3 Click the Add a Component Property (+) button at the upper left of the Schema tab to add a component property.

4 In the Schema Attributes pane (the bottom pane of the Schema tab), enter **editable** for the field name value and select Boolean for the data type value.

5 Click the Bindings tab and click the Add Binding (+) button to add a binding.

6 In the Add Binding dialog box, select the `editable` property, and click OK.

7 In the Binding Attributes pane at the bottom of the Bindings tab, click the Bound To item under Name, and click the magnifying glass icon across from the Bound To item under Value.

8 In the Bound To dialog box, under Component Path, select the CheckBox component, and then click OK.

9 Select the Checkbox component on the Stage, and click the Parameters tab in the Component Inspector panel.

10 Select Control > Test Movie. To test the functionality, type a value into the TextInput component and then deselect the CheckBox component. You should now be unable to enter text into the TextInput component.

Schema item settings

The schema of a component shows you what properties and fields are available for data binding. For each property or field, there are settings that control validation, formatting, type conversion, and other features that affect how data binding and the data management components handle the data of a field. The Schema Attributes pane, the bottom pane of the Schema tab, presents these settings, which you can view and edit. Settings fall into five groups, according to the features they control:

Basic settings Every field or property has these basic schema settings. In many cases, these are the only settings you need to bind to a field.

- Name: Every field needs a name.
- Data Type: Every field has a data type, which is selected from a list of available data types. The data type of a field affects data binding in two ways: When a new value is assigned to a field through data binding, the data type determines the rules that are used to check the data for validity. When you bind between fields that have different data types, the data binding feature attempts to convert the data appropriately.
- Storage Type: Every field has a storage type. Typically, it is defaulted to one of four values based on the data type of a field. The available values for storage types are simple, attribute, array, or complex.

  Note: Developers will almost never have to change this setting. However, there are some cases when the storage type for an attribute contained within the schema for an XML file may be defaulted to scalar when it should be set to attribute.

- Path (optional): If you are creating a virtual schema, you can use this property to identify the location of the data for this schema field.

Validation settings Validation settings are applicable to any field that is the destination of a binding. You will usually modify these settings when you want to control the validation of data that is input by the end user. To set this up, you bind from the UI component to a data component, and then select appropriate validation settings for the fields of the data component. One common example is when the user input is bound to the `params` property of a connector component, such as the XMLConnector component or WebServiceConnector component. Another common example is when UI components are bound to data fields of the DataSet component.

Validation operates in the following way: After any binding is executed, the new data is checked according to the validation rules of the destination field's data type. A component event is then generated to signal the results of the checking. If the data is found to be valid, then the valid event is generated; otherwise, an invalid event is generated. Both components involved in the binding emit the event. You can ignore these events. If you want anything to happen as a result of these events (such as giving feedback to the end user), you must write some ActionScript code that receives the valid and/or invalid events.

- Validation Options: Validation options are additional settings that affect the validation rules for this field. The settings are presented in the Validation Options dialog box, which appears when you select this item. These settings vary according to data type. For example, the String data type has settings for the allowed minimum and maximum length of the data.
- Required: This is a Boolean value that determines whether this field is required to have a non-null value. Validation fails if `required=true` but no value has been set.
- Read-Only: This is a Boolean value that determines whether this field is allowed to receive new values through data binding. If `readonly=true`, then executing any binding to this field generates the invalid event, and the field is changed.

Formatter settings Formatter settings are applied when a field's value needs to be converted to a string. Most often this will be for display purposes, such as when a DataSet field is databound to the `text` property of a Label or TextArea component. Formatter settings on a field are ignored when that field is databound to something whose data type isn't String.

- Formatter: The name of the formatter to use when converting this field to String. This is selected from a list of available formatters.
- Formatter Options: these are additional settings that affect the formatter. The settings are presented in the Formatting Options dialog box, which appears when you select this item. These settings vary according to formatter. For example, the Boolean formatter has settings for the text that represents the `true` and `false` values.

Note: If you don't specify a formatter, then a default conversion is applied when a field's value is needed as a string.

For a complete list of formatters, see "Schema formatters (Flash Professional only)" on page 269.

Kind and Encoder settings The Kind and Encoder settings are used to activate certain special features. For information, see "Using kinds and encoders" on page 263.

- Kind: The Kind setting for this field.. This is selected from a list of available Kind settings.
- Kind Options: Additional settings that affect the Kind setting. The settings are presented in the Kind Options dialog box, which appears when you select this item. These settings vary according to kind.
- Encoder: The Encoder setting for this field. This is selected from a list of available Encoder settings.
- Encoder Options: Additional settings that affect the encoder. The settings are presented in the Encoder Options dialog box, which appears when you select this item. These settings vary according to encoder.

Default settings These settings let you set defaults for various situations. There are two uses for these settings:

- If a field's value is undefined, then the default value is used instead whenever the value of the field is used as the source of a data binding. For example, the data fields of a DataSet component, or the `results` property of a connector component, are allowed to have an undefined value.
- When you create a new row of data in a DataSet component, the default value is used as the value of newly created records

When to edit schema item settings

When you build an application using data components and/or data binding, you need to apply schema item settings to some, but not necessarily all, fields of the components in your application. The following table summarizes the most common uses of schema item settings and will help you determine when these settings need to be edited.

Component	Property/field	Settings	When to use
Any connector	`params` (and its subfields)	Validation Options, Read-Only, Required	If validation is desired
	`results` (and its subfields)	Formatter, Formatter Options	For fields that need formatting for display as text
		Default value	For fields whose value is sometimes undefined
DataSet	Any data field	Name, Data Type	You must set these for every data set field that you define
		Validation Options, Read-Only, Required	If validation is desired
		Formatter, Formatter Options	For fields that need formatting for display as text
		Default Value	For fields whose value is sometimes undefined, or to specify the initial value for newly created data set records

Component	Property/field	Settings	When to use
UI components	UI components typically don't need any changes to their schema settings		
Any component	Any property or field	Kind, Kind Options, Encoding, Encoding Options	Various purposes, as described in "Using kinds and encoders" on page 263
Any connector	`results` (and its subfields)	Path	To identify the location of the data for a virtual schema field

Using kinds and encoders

Kinds and encoders are drop-in modules that perform additional special processing of the data of a schema item. They are often used in conjunction with each other to accomplish common tasks. Below is a list of common uses for kinds and encoders.

Calculated DataSet Fields Calculated fields are virtual fields that do not exist in the underlying data tables. Calculated fields provide developers with the ability to create and update dynamic field values at runtime. This is especially convenient for calculating and displaying values based on calculations or concatenations performed other fields located in a record (for instance, you can create a calculated field that combines the first and last name fields together to display the full name to a user).

To set up calculated fields for the DataSet component:

1. Select the DataSet component and click the Schema tab of the Component Inspector.
2. Create a persistent field for the DataSet component using the Add a Component Property (+) button.
3. Using the Schema Attributes pane, give the new component property a field name and set its kind to calculated.
4. Use the `calcFields` event of the DataSet component to assign this field a value at runtime.

   Note: You should assign a value to a calculated field only within the DataSet component's `calcFields` event.

Setting up schemas for XML documents In an XML document, all data is stored as a string. Sometimes you want the fields of an XML document to be available as data types other than String. The following example shows an application that pulls in data from an XML file.

```
<datapacket>
   <row id="1" billable="yes" rate="50" hours="3" />
   <row id="2" billable="no" rate="50" hours="6" />
</datapacket>
```

If you use this XML file to import a schema for the XMLConnector component's `results` property, it generates the following:

```
results : XML
   datapacket : Object
      row : Array
         [n] : object
            @billable: Boolean
            @hours : Integer
            @id : Integer
            @rate : Integer
```

But suppose you want to treat the row node as a record within a grid, and you want the `@billable` attribute to be treated as a Boolean value and display a `true` or `false` value in the grid instead of a `yes` or `no`. Getting the data into the grid is simple. You can simply bind the row schema field to the `dataProvider` property of the grid. The following procedure explains how to get the `@billable` attribute to be treated as a Boolean value and display a `true` or `false` value.

To make the @billable attribute display a true or false value:

1 Select the XMLConnector component and click the Schema tab.

2 Select the @billable schema field and select the `encoder` property in the Schema Attributes pane.

3 Make sure Encoder is set to Boolean. Select Encoder Options. You now have a choice to define which strings represent a `true` value and which strings represent a `false` value.

4 Enter **yes** for strings that you want to set to `true` and enter **no** for strings that you want to set to `false`.

The encoder now takes the XML data in its raw form (String) and converts it into its normal form (ActionScript type of Boolean). Using the encoder options, it knows how to encode the string values correctly.

5 Click Formatter and select Boolean. Select Formatter Options. You now have a choice to define how a `true` and `false` value should be displayed as a string.

6 Enter **True** for strings that mean `true` and enter **False** for strings that mean `false`.

The formatter now takes the normal form (ActionScript type Boolean) and formats it into the String value `true` or `false` for display purposes.

Setting the schema path

The `path` property for a schema field is an optional setting that is used in special circumstances when the schema for your component is not appropriate. Using this setting, you can create a virtual schema field (a field that exists in one location but pulls its value from another). The value of this property is a path expression that is entered in one of the following formats:

- For schemas that contain ActionScript data, the path follows the format `field [.field]...`, where `field` is equal to the name of a field (such as `addresslist.street`).
- For schemas that contain XML data, the path follows the format `XPath`, where `XPath` is a standard XPath statement (such as `addressList/street`).

When data binding is performed, Flash checks to see if there is a path expression for a schema field. If there is, it uses the path expression to locate the correct value. See "Virtual XML schemas" on page 240.

Note: The path expression is always performed relative to the parent node of the schema field.

About editing the contents of the Schema Attributes pane

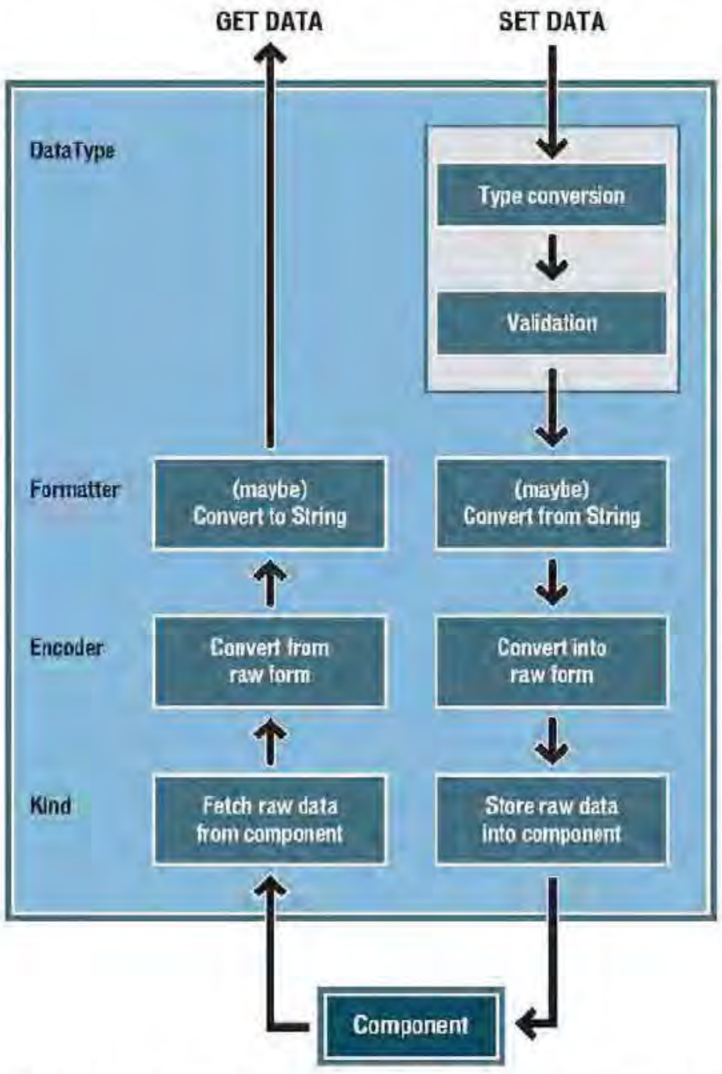
You can edit anything within the Schema Attributes pane—even schemas that came from an external source, such a web service WSDL file. You can always change any value for any field of any schema, with the following restrictions:

- If you change the type, then all the other schema item attributes are reset to the default values for the new data type.
- If you choose to completely reload the schema for a component property, you will lose all the edits that had previously been made within the Schema Attributes pane.

Note: There are several ways to reload the schema for a component property, including entering a new WSDL URL, choosing a different operation for a web service, or importing a new XML schema from a sample XML file.

How schema settings work (Flash Professional only)

Every schema item has four attributes: Data Type, Encoder, Formatter, and Kind. These settings control how data binding processes data values that it gets from, and sets to, components.



How schema settings modify the data flowing in and out of a component

When Flash wants to get data from a component, the data is fetched from the component according to the Kind setting. At this point, the data is in whatever format the component provides (the raw form of the data). For example, the XMLConnector component always provides data as a string, the NumericStepper component provides nNumbers, and so on. The encoder's job is to convert this data to an ActionScript data type; for example, the string data that you get from an XML document may really represent a date or a number (the normal form of the data). If data binding needs the data in string form (because it is being assigned to a text component, for example) the formatter is used to do this conversion. If there are several bindings from a field, the formatter is used only for those bindings that are assigning to a field whose type is String.

When you want to set data into a component, you first need to convert the data to the normal form—this conversion is automatic, depending on the Data Type setting. If the data is a string and a Formatter setting exists, then the formatter is used to convert the data from string to normal form. The Data Type setting also controls whether you inspect the data to see if it's valid, and returns valid or invalid events accordingly. The encoder is then used to convert the data from normal to raw form, and the kind then finally passes the data to the component.

This processing takes place only when the data field is accessed via data binding. It is possible to directly access a component property from your Actionscript code, but when you do this, you're working with the raw value of the data, not the data value that results from the action of data types, encoders, formatters, and kinds.

Guidelines for changing the schema item settings from their default values (Flash Professional only)

The following guidelines specify when to change the schema item settings from their default values:

- You always need a kind. However, if you leave `Kind = "none"`, it is the same as saying `Data`, which is usually the preferred setting.
- You need an encoder when the component does not provide the data in the form you want. The most common case is the XMLConnector component, or any other component whose properties are XML data. This is because XML stores all data—including numbers, dates, and Boolean values—as strings. So if you want to use the actual data (the number, Boolean value, or date) instead of the string representation of the data, you use an encoder.
- You need a formatter when you want to control how the data is converted to a string, usually for display purposes.
- You need a data type when you want data validation to occur, you want better conversion for certain data types, or both.

Schema kinds (Flash Professional only)

A kind determines how a schema item for your component should be accessed at runtime. The number of kinds allowed is unlimited, and you can create additional custom kinds. Kinds are defined by XML files found in the Flash MX Professional 2004 Configuration/Kinds folder. The definition includes the following metadata:

- An ActionScript class that will be instantiated to mediate access to the data
- A Kind Options dialog box

The following kinds ship with Flash MX Professional 2004:

None The default kind. This kind is identical to the Data kind.

Data The schema item is a data structure, and the data field is stored within the data structure as specified by the field's schema location. This is the normal case. The data structure can be in either ActionScript or XML form.

Calculated This kind is used in conjunction with the DataSet component. It is used to define a calculated field (a virtual field whose value is calculated at runtime). There is no special processing when getting or setting the value of a calculated field. For example, in the DataSet component you might define three fields, called `price`, `quantity`, and `totalPrice`. You would set the kind property for `totalPrice` to `Calculated` so that you can assign it a value at runtime, as in the sample code below:

```
function calculatedFunct(evt) {
      evt.target.totalPrice = (evt.target.price * evt.target.quantity);
    }
    ds.addEventListener('calcFields', calculatedFunct);
}
```

Schema encoders (Flash Professional only)

An encoder determines how a schema item for your component should be encoded/decoded at runtime. The number of encoders allowed is unlimited, and you can create additional custom encoders/decoders. Encoders are defined by XML files found in the Flash MX Professional 2004 Configuration/Encoders folder. The definition includes the following metadata:

- An ActionScript class that will be instantiated to encode/decode the data. This class must be a subclass of mx.databinding.DataAccessor.
- An Encoder Options dialog box

The following encoders ship with Flash MX Professional 2004:

None The default encoder. No encoding/decoding is performed.

Boolean Converts string data into Boolean ActionScript types. You must specify (via the Encoder Options property) one or more strings that will be interpreted as `true`, and one or more strings that will be interpreted as `false`. The settings are case-sensitive.

Date Converts string data into date ActionScript types. You must specify (via the Encoder Options property) a template string, which works as follows:

- The template string should contain 0 or 1 instances of `"YYYY"`, `"MM"`, `"DD"`, `"HH"`, `"NN"`, and/or `"SS"`, mixed with any other combination of characters.
- When converting from date to string, the numeric year, month, date, hour, minutes, and seconds, respectively, are substituted into the template, in place of YYYY, MM, and so on.
- When converting from string to date, the string must *exactly* match the template, with the correct number of digits for each of year, month, day, and so on.

DateToNumber Converts a Date object into its numeric equivalent. The DataSet component uses this encoder for fields that are of type Date. These values are stored within the DataSet component as numbers so that they can be sorted correctly.

Number Converts string data into number ActionScript types. There are no authoring settings for this encoder.

DatasetDeltaToXUpdateDelta This encoder is used in conjunction with the DataSet component. It extracts information from a DeltaPacket and generates XPath statements that are passed to the XUpdateResolver component to generate XUpdate statements. It gets the information that is needs to generate the XPath statements from two places:

- The `rowNodeKey` property, which you must specify via the Encoder Options property (defined below).
- Within the schema that was used for the XMLConnector component that originally retrieved the data.

Using this information, the encoder can generate the correct XPath statements needed to identify your data within the XML file.

The encoder options contain one property:

- The `rowNodeKey` property (String type). In order for an XML file to be updatable, the file must be structured in such a way that the node that represents a record in your data set can be uniquely identified with an XPath statement. This property combines an XPath statement with a field parameter to uniquely identify the row node within the XML file and the field within the data set that makes it unique.

  In the following example, the row node represents a record within the XML file. The value of the `id` attribute is what makes the row unique.

  ```
  <datapacket>
     <row id="1" date="01/01/2003" rate="50" hours="5" />
     <row id="2" date="02/04/2003" rate="50" hours="8" />
  </datapacket>
  ```

  The XPath to uniquely identify the row node would be as follows:

  ```
  datapacket/row[@id='xxx']
  ```

  Here, xxx represents a value for the `id` attribute. In a typical case, the `id` attribute in the XML file would be bound to the `id` field of the DataSet component. Therefore, the `rowNodeKey` value would be as follows:

  ```
  datapacket/row[@id='?id']
  ```

  The ? identifies that this is a field parameter. The `id` value specifies the name of the field in the data set. At runtime, the XUpdateResolver component substitutes the value from the `id` field of the data set to generate the correct XPath for the record in question.

In the next example, the contacts node with a category attribute of "Management" represents the record(s) within the XML file, and the employeeId subnode contains the value that makes the record unique:

```
<datapacket>
   <company id="5" name="ABC tech">
      <contacts category="Mgmt">
        <contact>
          <empId>555</employeeId>
          <name>Steve Woo</name>
          <email>steve.woo@abctech.com</email>
        </contact>
        <contact>
          <empId>382</employeeId>
          <name>John Phillips</name>
          <email>john.phillips@abctech.com</email>
        </contact>
        ...
        ...
      </contacts>
      <contacts category="Executives">
        ...
        ...
      </contacts>
      ...
      ...
   </company>
</datapacket>
```

The `rowNodeKey` value for this XML file would be as follows:

```
datapacket/company/contacts[@category='Mgmt']/contact[empId='?empId']
```

Schema formatters (Flash Professional only)

A formatter is an object that performs bidirectional conversion of data between a raw data type and string data. The object has parameters that are settable during authoring, and runtime methods for doing the conversion. The number of formatters allowed is unlimited, and you can create additional custom formatters. Formatters are defined by XML files found in the Flash MX Professional 2004 Configuration/Formatters folder. The definition includes the following metadata:

- The ActionScript class that will be instantiated to perform the formatting
- A Formatter Options dialog box

The following formatters ship with Flash MX Professional 2004:

None The default formatter. No formatting is performed.

Boolean This formatter formats a Boolean value as a string. You can set up Boolean options for strings that mean `true` (e.g., 1, yes, yup) and strings that mean `false` (e.g., 0, no, nope).

Compose String This formatter converts a data object to a string. You define the output format using a string template. The template is arbitrary text that can refer to fields of the data as one of the following:

- `<field-name>`
- `<field-name.field-name>`, using dots to drill down into the data structure
- `<.>`, which represents the entire object. This can be used, for example, when the original object is a string, in which case `<.>` is simply the value of the string.

Custom Formatter This formatter allows you to specify your own formatter by specifying a class name. The formatter ActionScript class should have the following format:

```
class MyFormatter extends mx.data.binding.CustomFormatter{
function getTypedValue(requestedType: String) : mx.data.binding.TypedValue{
...
}
function getGettableTypes() : Array /* of String */{
...
}
function setTypedValue(newValue: mx.data.binding.TypedValue) : Array /* of
  String */{
...
}
function getSettableTypes() : Array /* of String */{
...
}
}
```

Rearrange Fields This formatter creates a new array of objects based on the original array in your binding. You define the fields on the new array by using a string template in the form:

```
fieldname1=definition1;fieldname2=definition2;and so on.
```

The `fieldnameN` are the names of the fields in the new array or records. The `definitionN` is one of the following:

- The name of a field in the original record
- A string, enclosed in single quotes, that contains a mix of text and tags. A tag is the name of a field in the original array, enclosed in < and >.
- A single dot ".", which represents the entire original record

For example, suppose you want to assign an array to the `DataProvider` property of a List component using data binding. The objects within the array do not have a `label` property (which the list uses if available). You could use this formatter to create a new array through data binding that replicates the objects within your original array and adds a `label` property to each object using the values you define. The following template would achieve this (this would be on a binding between your array and the List component's `DataProvider` property):

```
label='My name is <firstName> <lastName>'
```

This syntax assumes that the object has two properties, called `firstName` and `lastName`. The `label` property will be added to each object within the new array.

Note: This formatter can be used with any component that accepts an array of records as a binding.

Number Formatter This formatter allows you to specify the number of fractional digits that appears when a number is converted to text.

Schema data types (Flash Professional only)

A data type is an object that represents all the runtime logic needed to support a particular data type. A data type can be a scalar type, such as int, string, date, currency amount, or zipcode. It can also be a complex type, with subfields and so on. A data type can test a data value to determine if it is valid for that data type. The number of data types allowed is unlimited, and you can create additional custom data types. Data types are defined by XML files found in the Flash MX Professional 2004 Configuration/DataTypes folder. The definition includes the following metadata:

- An ActionScript class that will be instantiated for validation and type conversion
- A Validation Options dialog box
- The name of the standard formatter, which you can override using the `formatter` property
- Initial values for required, read-only, and default values

The following data types ship with Flash MX Professional 2004:

Note: These data types are able to perform validation: Custom, Integer, Number, PhoneNumber, SocialSecurity, String, ZipCode. These data types are able to convert from various other data types when you assign to them: Boolean, DataProvider, Integer, Number, String, XML.

Array Array data type. There are no validation options.

Attribute XML attribute. There are no validation options.

Boolean Boolean data type.There are no validation options.

Custom Allows you to add your own custom class to check for this special kind of validation. The Actionscript class has to be in the classpath and formatted as follows:

```
class myCustomType extends mx.databinding.CustomValidator {
  function validate(value) {
      ... some code here
    }
}
```

DataProvider DataProvider data type. There are no validation options.

Integer Integer data type. A validation option can be set up to define the minimum and maximum range for the integer value.

Number Number data type. Like Integer, this option allows you to set up a range for minimum and maximum number values.

Object There are no validation options.

PhoneNumber There are no validation options.

SocialSecurity There are no validation options.

String This option allows you to set up the minimum and maximum number of characters for a string value.

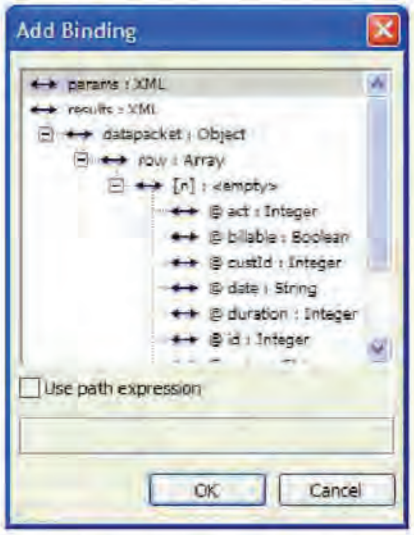
XML There are no validation options.

ZipCode There are no validation options.

Working with bindings in the Bindings tab (Flash Professional only)

The Bindings tab allows you to view, add, and remove bindings. All the bindings for a component are displayed here. The Binding List pane, the top pane of the Bindings tab, displays a component's bound properties, represented by their schema location. The Binding Attributes pane, the bottom pane, displays name/value pairs for the selected binding.

When you click the Add Binding (+) button on the Bindings tab, the Add Binding dialog box appears.



This dialog box displays all the schema items for your component. You use this dialog box to select which schema item to bind to. Component properties are displayed as root nodes within the schema tree. An arrow icon represents whether a schema item has read/write access, as follows: a right-pointing arrow represents a write-only property, a left-pointing arrow represents a read-only property, and a bidirectional arrow represents a read-write property.

Entering a path expression

The schema for a component defines which schema items are bindable. However, you may find a need to bind to a schema item that is not identified in the schema tree. You do this by entering a path expression.

To enter a path expression:

1 In the Bindings tab of the Component Inspector panel, click the Add Binding (+) button.

2 In the Add Binding dialog box, select Use Path Expression.

Only the component property names appear in the schema tree when the option is selected.

3 Select a component property from the tree for which you want to create an expression. Once a component property is selected, the Path Expression text box becomes active.

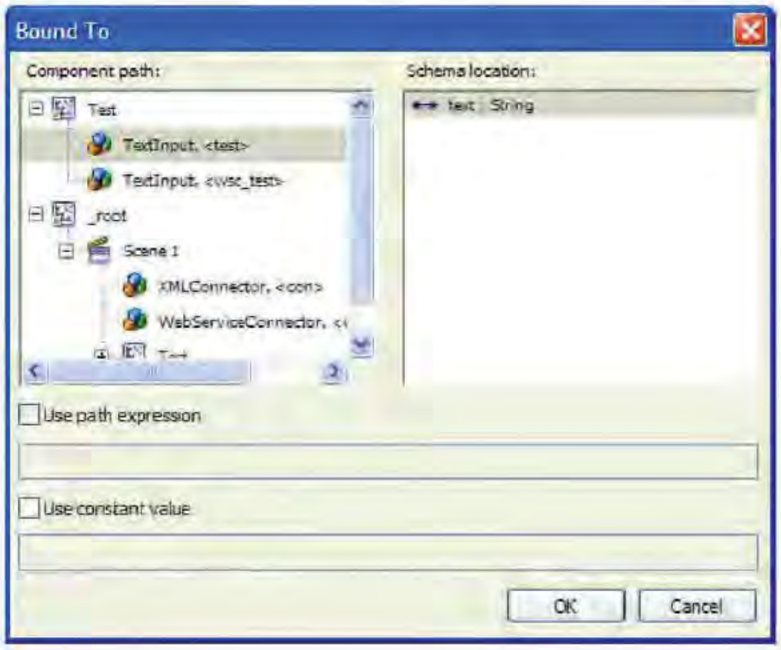
4 Enter a path expression to identify the schema item that you want to bind to. Path expressions are entered in the following formats:

- For properties that contain ActionScript data, the path follows this format:
  `field [.field]...`
  where `field` is equal to the name of a field (such as `addresslist.street`).
- For properties that contain XML data, the path follows this format:
  `XPath`
  where `XPath` is a standard XPath statement (such as `addressList/street`).

5 Click OK to return to the Bindings tab.

Using the Bound To dialog box

The Bound To dialog box appears when you click Bound To in the Binding Attributes pane of the Bindings tab. The Bound To dialog box includes the Component Path pane and the Schema Location pane.



JPMC-00157869

The Component Path pane shows a tree of components that have bindable properties. The tree is based on the current Stage editing environment:

- If the Stage is displaying the contents of the document root, a single component path tree is displayed relative to the document root.

  Note: Component instances are displayed only if they exist in Frame 1 of the edited document root or in Frame 1 of any screen/clip whose instance exists in the edited document root. This pane shows only components, not text fields.

- If the Stage is displaying the contents of a movie clip being edited from the library, two component path trees are displayed. The first is displayed from the root of the symbol being edited, and the second is displayed from the document root, to allow bindings to instances within the document.

  Note: Bindings to this second component tree are not displayed in the Bound To instances when they are selected. They appear only as bindings from the Bound From component instance.

The Schema Location pane shows the schema tree of the component selected in the Component Path pane.

Note: This is the same information that appears in the Schema Tree pane of the Component Inspector panel's Schema tab.

You can use a dynamic value or a constant value for the Bound To property.

To use a dynamic value for the Bound To property:

1 Select a component in the Component Path pane.

2 Do one of the following to select a schema item for the data:

- Select a schema item using the Schema tree located within the Schema Location pane.
- Select Use Path Expression and then select a component property from the schema tree and enter a path expression. See "Entering a path expression" on page 273.

To use a constant value for the Bound To property:

- Select Use Constant Value and enter a constant value, such as 3, a string, or `true`. You can use any value that is valid for the schema item. When you use a constant value, the selected component path, schema location, and path expression are all ignored. You can bind to a constant value only when the Direction attribute for the binding is set to In.

Binding attributes

When a binding is selected within the binding list, you can further define it using the properties located within the Binding Attributes pane, the bottom pane of the Bindings tab. From here, you can specify additional information, such as Direction, Bound To, Formatter, and so on. The Binding Attributes pane contains several properties that apply to all bindings:

Direction Displays a list of directions that can be set for a binding:

- In: The selected schema item is the destination of a binding. It receives a new value when the other end of the binding changes.
- Out: The selected schema item is the source of a binding. Whenever its value changes, the value is copied to the other end of the binding.
- In/Out: New data values are copied when either end of the binding changes value.

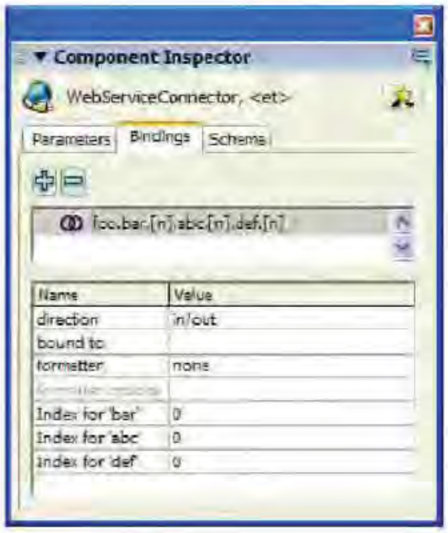
Bound To Identifies the destination schema item (another component's schema item) that this schema item is bound to.

Formatter Displays a list of available formatters that determine how to display this binding. See "Schema formatters (Flash Professional only)" on page 269.

Formatter Options Displays the Formatting Options dialog box. The settings in this dialog box are used at runtime to control formatting of data assigned from this schema item to the destination schema item that is defined in the Bound To property. These settings override the default formatting settings for the source schema item.

Index For If you are creating a binding for a schema item that is defined as a field of an object contained within an array, you must specify an index for the array (that is, if the location of the schema item is `results.Theaters[].PlayingMovies`, an index must be defined for `Theaters[]`). In this situation, a new setting is dynamically added to the Binding Attributes pane.

Note: If a schema item location includes several array references such as `"foo/bar[]/abc[]/def[]"`, three `index for` settings are dynamically added to the Schema Attributes pane—one for each array that needs to be indexed.



foo.bar.[n].abc[n].def.[n]

Name	Value
direction	in/out
bound to	
formatter	none
formatter options	
Index for 'bar'	0
Index for 'abc'	0
Index for 'def'	0

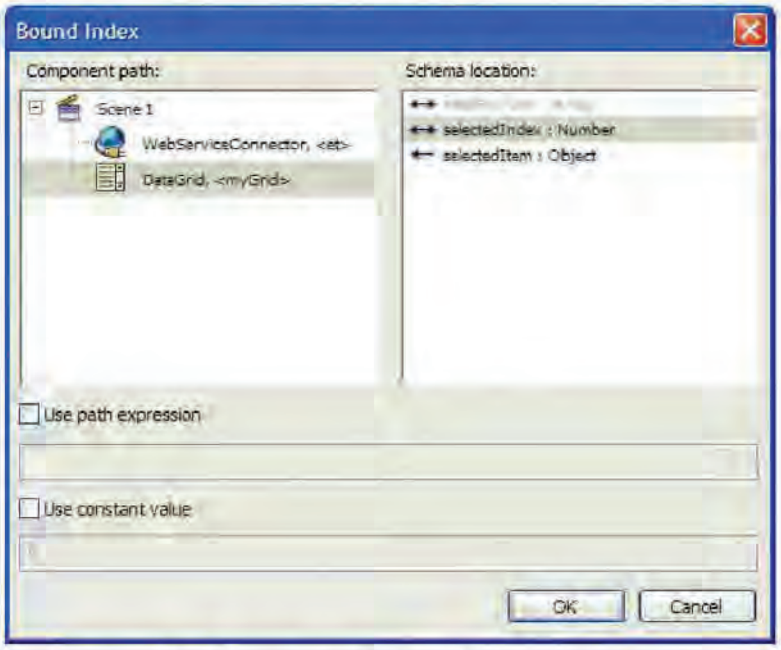
To define an index:

1 Select the `index for` setting in the Binding Attributes pane.

2 Click the magnifying glass icon in the Value column.

The Bound Index dialog box appears. This dialog box works in the same way as the Bound To dialog box. See "Using the Bound To dialog box" on page 273.

3 Use the Bound Index dialog box to define a dynamic index value (such as the `selectedIndex` property of a DataGrid component) or constant value as an index for your array. If you use a dynamic index value, the binding is reperformed each time the dynamic index value changes.



In this example, the user is selecting the `selectedIndex` property of the DataGrid component as the index. Each time the end user selects a new record in the data grid, the index of the array is updated to show the data associated with the new record.

Note: The `index for` property appears only in the binding attributes for the schema item that is the field of the object within the array. If you select the Bindings tab for the DataGrid component, the `index for` property does not appear in the Binding Attributes pane for the selected index schema item.

Sometimes you might need to manually define a schema that identifies a schema item as a field of an object contained within an array. In the following example, the @id, @billable, @rate, and @duration schema fields are all considered fields of and object contained within the row array:

```
results : XML
   datapacket : Object
      row : Array
         [n] : object
            @id : Integer
            @billable : Boolean
            @rate : Number
            @duration : Integer
```

If a binding is created for any of these items, an `index for 'row'` property appears in the Binding Attributes pane, so that an index can be specified for the row array. The authoring environment uses the [n] schema field to identify this type of relationship. Therefore, you may need to duplicate this entry if you are manually creating a schema. To do this, you add a new schema field under the "row : Array" node and set Field Name for the schema field to `[n]`. The authoring tool reads this value and create an `index for` property if it is used within a binding.

About debugging data binding and web services (Flash Professional only)

Data binding is a series of actions that take place in response to events, such as the following:

- The data of a component property changes.
- A web service call is completed.
- An XML document is fetched.

All actions that are performed by data binding or web services are reported to a log, which you can control by turning the trace log on:

```
_global.__dataLogger=new mx.data.binding.Log();  //to enable trace log
```

Or you can turn the trace log off:

```
_global.__dataLogger=null; //disable trace for binding.
```

When you run an application that turns the trace on, a detailed log of data binding and web services events and actions appears in the Output window. The following are the types of things reported:

- Executing bindings
- Calling web service methods
- Fetching XML documents
- Status and result events from WebService and XML components
- Valid and invalid events from validated data fields
- A variety of errors, invalid settings, and so on that are checked for

By running your application and then examining the log, you can often find out why things are not working as expected. Sometimes an error is explicitly reported—for example, a missing web service parameter. Sometimes the data is bound to the wrong component, or to no component, and so on. If you find that there is too much information in the log, clear the Output window by selecting Clear from the context menu, to keep the log as short as possible.

Working with data binding and web services at runtime (Flash Professional only)

You can work with data binding and web services at runtime without using any of the authoring features, such as the Binding and Schema panels in the Component Inspector panel or the WebServiceConnector component. You can do this by creating instances of the DataBinding or WebService classes. For more information, see "Binding class" and "WebService class" in the "Components Dictionary" in Using Components Help.

The byte code for the DataBinding and WebService classes is contained within SWC files that must be present in your library in order to compile and generate your output SWF file. These SWC files are contained within the Classes common library.

To add the SWC files to your library:

1. Select the Classes library by selecting Windows > Other Panels > Common Libraries > Classes.
2. Open the library for your document by selecting Window > Library.
3. Drag the appropriate SWC file or files (DataBindingClasses and/or WebServiceClasses) from the Classes library into your document's library.

Note: This is the only way to get the needed classes into your document so that it will compile and run.

CHAPTER 15 Publishing

When you're ready to deliver your Macromedia Flash MX 2004 and Macromedia Flash MX Professional 2004 content to an audience, you can publish it for playback. By default, the Publish command creates a Flash SWF file and an HTML document that inserts your Flash content in a browser window. The Publish command also creates and copies detection files for Flash 4 and later. If you change publish settings, Flash saves the changes with the document. You can create publish profiles to name and save various configurations for the Publish Settings dialog box, in order to quickly publish documents a variety of ways. After you create a publish profile, you can export it for use in other documents, or for use by others working on the same project. See "Using publish profiles" on page 295.

If you're publishing content that targets Flash Player 4 or later versions, you can implement Flash Player detection, which checks your user's version of Flash Player. If the user doesn't have the specified version, you can direct the user to an alternate web page. See "Configuring publish settings for Flash Player detection" on page 287

The Macromedia Flash Player 6 and later versions support Unicode text encoding. With Unicode support, users can view multilanguage text, regardless of the language used by the operating system running the player. See Chapter 13, "Creating Multilanguage Text," on page 215.

You can also publish the FLA file in alternative file formats—GIF, JPEG, PNG, and QuickTime—with the HTML needed to display them in the browser window. Alternative formats allow a browser to display your SWF file animation and interactivity for users who don't have the targeted Flash Player installed. When you publish a FLA file in alternative file formats, the settings for each file format are stored with the FLA file.

You can export the FLA file in a variety of formats as well. Exporting FLA files is similar to publishing FLA files in alternative file formats, except that the settings for each file format are not stored with the FLA file. See Chapter 16, "Exporting," on page 311.

As an alternative to using the Publish command, if you're proficient in HTML, you can create your own HTML document with any HTML editor and include the tags required to display a SWF file. See "About configuring a web server for Flash" on page 310.

Before you publish your SWF file, it's important to test how the SWF file works using the Test Movie and Test Scene commands.

Playing your Flash SWF files

The Macromedia Flash SWF file format is for deploying Flash content.

You can play Flash content in the following ways:

- In Internet browsers such as Netscape Navigator and Internet Explorer that are equipped with Flash Player 7
- With the Flash Xtra in Director and Authorware
- With the Flash ActiveX control in Microsoft Office and other ActiveX hosts
- As part of a QuickTime movie
- As a stand-alone movie called a projector

The Flash SWF format is an open standard that is supported by other applications. For more information about Flash file formats, see the Macromedia website at www.macromedia.com/software/flashplayer/.

About publishing secure Flash documents

Flash Player 7 contains several features that help you ensure the security of your Flash documents.

Buffer overrun protection

Buffer overrun protection prevents the intentional misuse of external files in a Flash document to overwrite a user's memory or insert destructive code such as a virus. This prevents a Flash document from reading or writing data outside the document's designated memory space on a user's system. Buffer overrun protection is enabled automatically.

About exact domain matching for sharing data between Flash documents

Flash Player 7 enforces a stricter security model than previous versions of Flash Player do. Between Flash Player 6 and Flash Player 7, there were two primary changes in the security model:

Exact domain matching Flash Player 6 lets SWF files from similar domains (for example, `www.macromedia.com` and `store.macromedia.com`) communicate freely with each other and with other documents. In Flash Player 7, the domain of the data to be accessed must match the data provider's domain *exactly* in order for the domains to communicate.

HTTPS/HTTP restriction A SWF file that loads using nonsecure (non-HTTPS) protocols cannot access content loaded using a secure (HTTPS) protocol, even when both are in exactly the same domain.

For more information about ensuring that Flash content performs as expected with the new security model, see "Flash Player security features" in ActionScript Reference Guide Help.

Publishing Flash documents

To publish a Flash document, you first select publish file formats and then select file format settings with the Publish Settings command. Then you publish the Flash document using the Publish command. The publishing configuration that you established in the Publish Settings dialog box is saved with the document. You can also create and name a publish profile so that the established publish settings are always available.

Depending on the options you specify in the Publish Settings dialog box, the Publish command creates the following files:

- The Flash SWF file
- Alternate images in a variety of formats that appear automatically when Flash Player is not available (GIF, JPEG, PNG, and QuickTime)
- The supporting HTML document(s) required to display SWF content (or an alternate image) in a browser and control browser setting
- Three HTML files (if you keep the default, Detect Flash Version, selected): the detection file, the content file, and the alternate file
- Stand-alone projector files for both Windows and Macintosh systems and QuickTime videos from Flash content (EXE, HQX, or MOV files, respectively)

Note: To alter or update a SWF file created with the Publish command, you must edit the original Flash document and then use the Publish command again to preserve all authoring information. Importing a Flash SWF file into Flash removes some of the authoring information.

For information on publish settings, see "Configuring publish settings for Flash Player detection" on page 287. For general information, see "Specifying publish settings that create HTML documents with embedded Flash content" on page 284.

To set general publish settings for a Flash document:

1 Open the Publish Settings dialog box. Do one of the following:
   - Select File > Publish Settings.
   - In the Property inspector for the document (which is available when no object is selected), click the Settings button.

   Note: To create a publish profile for the publish settings that you'll specify, see "Using publish profiles" on page 295.

2 In the Publish Settings dialog box, select the option for each file format you want to create.

   The Flash SWF format is selected by default. The HTML format is also selected by default, because an HTML file is needed to display a SWF file in a browser. Tabs corresponding to the selected file formats appear above the current panel in the dialog box (except for Windows or Macintosh projector formats, which have no settings). For more information on publish settings for individual file formats, see the sections that follow.

3 In the File text box for each selected format, either accept the default filename, which corresponds to the name of the document, or enter a new filename with the appropriate extension (such as .gif for a GIF file and .jpg for a JPEG file).

4 Choose where to publish the files. By default, the files are published in the same location as the FLA file. To change where files are published, click the folder beside the filename and browse to a different location in which to publish the file.

5 To create a stand-alone projector file, select Windows Projector or Macintosh Projector.

Note: The Windows version of Flash adds the .hqx extension to the filename of a Macintosh projector file. You can create a Macintosh projector using the Windows versions of Flash, but you must use a file translator such as BinHex to make the resulting file appear as an application file in the Macintosh Finder.

6 Click the tab for the format options you want to change. Specify publish settings for each format, as described in the following sections.

7 When you have finished setting options, do one of the following:

- To generate all the specified files, click Publish.
- To save the settings with the FLA file and close the dialog box without publishing, click OK.

To publish a Flash document without choosing new publish settings:

- Select File > Publish to create the files in the formats and location specified in the Publish Settings dialog box (either the default settings, the settings you selected previously, or the selected publish profile).

Setting publish options for the Flash SWF file format

When publishing a Flash document, you can set image and sound compression options, and an option to protect your SWF file from being imported. You use the controls in the Flash panel of the Publish Settings dialog box to change the settings.

To set publish options for a Flash document:

1 Open the Publish Settings dialog box. Do one of the following:

- Select File > Publish Settings.
- In the Property inspector for the document (which is available when no object is selected), click the Settings button.

Note: To create a publish profile for the publish settings that you'll specify, see "Using publish profiles" on page 295.

2 Click the Flash tab and select a Player version from the Version pop-up menu.

Not all Macromedia Flash MX 2004 and Macromedia Flash MX Professional 2004 features work in published SWF files that target Flash Player versions earlier than Flash Player 7.

If you want to specific Flash Player detection, on the HTML tab of the Publish Settings dialog box, you must select Flash Player 4 or a later version. For more information about Flash Player detection, see "Configuring publish settings for Flash Player detection" on page 287.

3 Select a load order to specify how Flash loads a SWF file's layers for displaying the first frame of your SWF file: Bottom Up or Top Down.

This option controls which parts of the SWF file Flash draws first over a slow network or modem connection.

4 In the ActionScript Version pop-up menu, select either ActionScript 1.0 or 2.0 to reflect the version used in your document.

If you select ActionScript 2.0 and you've created classes, you can click the Settings button to set the relative classpath to class files that differs from the path to default directories set in Preferences. For more information, see "Setting the classpath" on page 284.

5 To enable debugging of the published Flash SWF file, select any of the following options:

Generate Size Report generates a report listing the amount of data in the final Flash content by file.

Omit Trace Actions causes Flash to ignore Trace actions (`trace`) in the current SWF file. When you select this option, information from Trace actions is not displayed in the Output panel.

For more information, see "Using the Output panel" in ActionScript Reference Guide Help.

Protect from Import prevents others from importing a SWF file and converting it back into a Flash (FLA) document. If you select this option, you can decide to use password protection with your Flash SWF file.

Debugging Permitted activates the Debugger and allows remote debugging of a Flash SWF file. If you select this option, you can decide to use password protection with your SWF file.

Compress movie compresses the SWF file to reduce file size and download time. This option is selected by default and is most beneficial when a file is text-intensive or includes a lot of ActionScript. A compressed file plays only in Flash Player 6 or later.

Optimize for Flash Player 6 r65 If you selected Flash Player 6 in the Version pop-up menu, you can select this option to target a release of Flash Player 6. The updated version uses ActionScript register allocation to improve performance. Users must have the same release of Flash Player 6 or later.

6 If you selected either Debugging Permitted or Protect from Import in step 5, you can enter a password in the Password text box. If you add a password, others must enter the password before they can debug or import the SWF file. To remove the password, clear the Password text box.

For more information on the Debugger, see "Writing and Debugging Scripts" in ActionScript Reference Guide Help.

7 To control bitmap compression, adjust the JPEG Quality slider or enter a value.

Lower image quality produces smaller files; higher image quality produces larger files. Try different settings to determine the best trade-off between size and quality; 100 provides the highest quality and least compression.

8 To set the sample rate and compression for all streaming sounds or event sounds in the SWF file, click the Set button next to Audio Stream or Audio Event and select options for Compression, Bit Rate, and Quality in the Sound Settings dialog box. Click OK when you are finished.

Note: A streaming sound plays as soon as enough data for the first few frames downloads; it is synchronized to the Timeline. An event sound is not played until it is downloaded completely, and it continues to play until explicitly stopped.

For more information on sound, see Chapter 11, "Working with Sound," on page 185.

9 If you want to use the settings selected in step 8 to override settings for individual sounds selected in the Sound section of the Property inspector, select Override Sound Settings. You may want to select this option to create a smaller low-fidelity version of a SWF file.

Note: If the Select Override Sound Settings option is deselected, Flash scans all stream sounds in the document (including sounds in imported video) and publishes all stream sounds at the highest individual setting. This can increase file size, if one or more stream sounds has a high export setting.

10 (Flash Professional only) To export sounds suitable for devices, including mobile devices, instead of the original library sound, select Export Device Sounds. For additional information, see "Using sounds in Flash documents for mobile devices (Flash Professional only)" on page 195. To save the settings with the current file, click OK.

Setting the classpath

To use an ActionScript class that you've defined, Flash must be able to locate the external ActionScript 2.0 files that contain the class definition. The list of folders in which Flash searches for class definitions is called the classpath. Classpaths exist at the global/application level, and at the document level. For more information about classpaths, see "Understanding the classpath" in ActionScript Reference Guide Help.

To modify the document-level classpath:

1 Select File > Publish Settings to open the Publish Settings dialog box.

2 Click the Flash tab.

3 Verify that ActionScript 2.0 is selected in the ActionScript Version pop-up menu and click Settings.

4 In the ActionScript Settings dialog box, specify the frame on which the class definition should reside in the Export Frame for classes text box.

5 Do any of the following:

- To add a folder to the classpath, click the Browse to Path button, browse to the folder you want to add, and click OK.

  Alternatively, click the Add New Path (+) button to add a new line to the Classpath list. Double-click the new line, type a relative or absolute path, and click OK.

- To edit an existing classpath folder, select the path in the Classpath list, click the Browse to Path button, browse to the folder you want to add, and click OK.

  Alternatively, double-click the path in the Classpath list, type the desired path, and click OK.

- To delete a folder from the classpath, select the path in the Classpath list and click the Remove from Path button.

Specifying publish settings that create HTML documents with embedded Flash content

Playing Flash content in a web browser requires an HTML document that activates the SWF file and specifies browser settings. This document is generated automatically by the Publish command, from HTML parameters in a template document.

The template document can be any text file that contains the appropriate template variables—including a plain HTML file, one that includes code for special interpreters such as ColdFusion or Active Server Pages (ASP), or a template included with Flash (for more information, see "About configuring a web server for Flash" on page 310).

You can customize a built-in template (see "Customizing HTML publishing templates" on page 297), or manually enter HTML parameters for Flash using any HTML editor (see "Editing Flash HTML settings" on page 301).

HTML parameters determine where the Flash content appears in the window, the background color, the size of the SWF file, and so on, and set attributes for the `object` and `embed` tags. You can change these and other settings in the HTML panel of the Publish Settings dialog box. Changing these settings overrides options you've set in your SWF file.

To publish HTML that displays your Flash SWF file:

1 Do one of the following to open the Publish Settings dialog box:

- Select File > Publish Settings.
- In the Property inspector for the document (which is available when no object is selected), click the Settings button.

Note: To create a publish profile for the publish settings that you'll specify, see "Using publish profiles" on page 295.

2 On the Formats tab, the HTML file type is selected by default. In the File text box for the HMTL file, either use the default filename, which matches the name of your document, or enter a unique name, including the .html extension.

3 Click the HTML tab to display HTML settings and select an installed template to use from the Template pop-up menu. Then click the Info button to the right to display a description of the selected template. The default selection is Flash Only.

4 If, in the previous step, you selected an HTML template other than Image Map or QuickTime, and on the Flash tab, you had the Version set to Flash Player 4 or later, you can select Flash Version Detection.

Note: Flash Version Detection configures your document to detect the version of Flash Player that the user has and sends the user to an alternate HTML page if the user does not have the targeted player. For more information on version detection, see "Configuring publish settings for Flash Player detection" on page 287.

5 Select a Dimensions option to set the values of the `width` and `height` attributes in the `object` and `embed` tags:

Match Movie (the default) uses the size of the SWF file.

Pixels enters the number of pixels for the width and height in the Width and Height field.

Percent specifies the percentage of the browser window that the SWF file will occupy.

6 Select Playback options to control the SWF file's playback and features, as follows:

Paused at Start pauses the SWF file until a user clicks a button or selects Play from the shortcut menu. By default, the option is deselected and the Flash content begins to play as soon as it is loaded (the `PLAY` parameter is set to `true`).

Loop repeats the Flash content when it reaches the last frame. Deselect this option to stop the Flash content when it reaches the last frame. (The `LOOP` parameter is on by default.)

Display Menu displays a shortcut menu when users right-click (Windows) or Control-click (Macintosh) the SWF file. Deselect this option to display only About Flash in the shortcut menu. By default, this option is selected (the `MENU` parameter is set to `true`).

Device Font (for Windows only) substitutes anti-aliased (smooth-edged) system fonts for fonts not installed on the user's system. Using device fonts increases the legibility of type at small sizes and can decrease the SWF file's size. This option affects only SWF files that contain static text (text that you created when authoring a SWF file and that does not change when the Flash content is displayed) set to display with device fonts. For more information, see "Using device fonts (static horizontal text only)" on page 104.

7 Select Quality options to determine the trade-off between processing time and appearance, as follows. This option sets the `QUALITY` parameter's value in the `object` and `embed` tags.

Low favors playback speed over appearance and does not use anti-aliasing.

Auto Low emphasizes speed at first but improves appearance whenever possible. Playback begins with anti-aliasing turned off. If Flash Player detects that the processor can handle it, anti-aliasing is turned on.

Auto High emphasizes playback speed and appearance equally at first but sacrifices appearance for playback speed if necessary. Playback begins with anti-aliasing turned on. If the actual frame rate drops below the specified frame rate, anti-aliasing is turned off to improve playback speed. Use this setting to emulate the View > Antialias setting in Flash.

Medium applies some anti-aliasing but does not smooth bitmaps. It produces a better quality than the Low setting, but lower quality than the High setting.

High (the default) favors appearance over playback speed and always uses anti-aliasing. If the SWF file does not contain animation, bitmaps are smoothed; if the SWF file has animation, bitmaps are not smoothed.

Best provides the best display quality and does not consider playback speed. All output is anti-aliased and bitmaps are always smoothed.

8 Select a Window Mode option, which controls the HTML `wmode` attribute in the `object` and `embed` tags. The window mode modifies the relationship of the Flash content bounding box or virtual window with content in the `HTML` page as follows:

Window does not embed any window-related attributes in the object and embed tags. The background of the Flash content is opaque and use the HTML background color. The HTML cannot render above or below the Flash content. Window is the default setting.

Opaque Windowless sets the background of the Flash content to opaque, obscuring anything underneath the Flash content. Opaque Windowless allows HTML content to be displayed above or on top of Flash content.

Transparent Windowless sets the background of the Flash content to transparent. This allows the HTML content to be displayed above and below the Flash content.

Note: In some instances, complex rendering in transparent windowless mode may result in slower animation when the HTML images are also complex.

See the table following this procedure for browsers that support windowless modes.

9 Select an HTML Alignment option to position the Flash SWF window in the browser window:

Default centers the Flash content in the browser window and crops edges if the browser window is smaller than the application.

Left, **Right**, **Top**, or **Bottom** align SWF files along the corresponding edge of the browser window and crop the remaining three sides as needed.

10 Select a Scale option to place the Flash content within specified boundaries, if you've changed the document's original width and height. The Scale option sets the `SCALE` parameter in the `object` and `embed` tags.

Default (Show All) displays the entire document in the specified area without distortion while maintaining the original aspect ratio of the SWF files. Borders may appear on two sides of the application.

No Border scales the document to fill the specified area and keeps the SWF file's original aspect ratio without distortion, cropping the SWF file if needed.

Exact Fit displays the entire document in the specified area without preserving the original aspect ratio, which may cause distortion.

No Scale prevents the document from scaling when the Flash Player window is resized.

11 Select a Flash Alignment option to set how the Flash content is placed within the application window and how it is cropped, if necessary. This option sets the `SALIGN` parameter of the `object` and `embed` tags.

- For Horizontal alignment, select Left, Center, or Right.
- For Vertical alignment, select Top, Center, or Bottom.

12 Select Show Warning Messages to display error messages if tag settings conflict—for example, if a template has code referring to an alternate image that has not been specified.

13 To save the settings with the current file, click OK.

The following browsers support windowless modes:

	Internet Explorer	Netscape	Other
Macintosh OS X 10.1.5 and 10.2	IE 5.1 and IE 5.2	7.0 and later	• Opera 6 or later • Mozilla 1.0 or later • AOL/Compuserve
Windows	IE 5.0, 5.5, and 6.0	7.0 and later	• Opera 6 and later • Mozilla 1.0 and later • AOL/Compuserve

Configuring publish settings for Flash Player detection

You can configure your document to detect your users' Flash Player version. If you've selected Detect Flash Version in the Publish Settings dialog box, users who access your Flash application are transparently directed to an HTML file that contains a SWF file designed to detect their Flash Player version. If they have the specified version or later, the SWF file again redirects the user to your content HTML file, and your SWF file plays as designed. If users don't have the specified version, they're redirected to an alternate HTML file that Flash creates, or that you've created.

To enable Flash Player detection:

1 Select Detect Flash Version on the HTML tab of the Publish Settings dialog box. For general information, see "Specifying publish settings that create HTML documents with embedded Flash content" on page 284.

   Note: The option is available only if you've selected Flash Player 4 or later on the Flash tab of the Publish Settings dialog box, and if you have not selected QuickTime or Image Map as a template.

2 Click Settings for Detect Flash Version. The dialog box shows the Flash Player version that you selected on the Flash tab of the Publish Settings dialog box. You can use the Major Revision and Minor Revision text boxes to specify precise revisions of Flash Player.

3 The Detection File text box displays the name of the HTML file that contains the SWF file designed to detect the Player version and redirect users to the appropriate HTML page. You can accept the default name, or change it.

   Note: Changing the default name also changes the name in the HTML text box on the Formats tab of the Publish Settings dialog box.

4 Use the Content File text box to specify the name of the HTML template that contains your Flash content. The default name is the name of your document, appended with _content.

5 Do one of the following to create the alternate HTML page, for users who don't have the specified Flash Player version:

   - If you'd like Flash to automatically create an alternate HTML file, select Generate Default and either accept the default filename in the Alternate File text box, or enter a new filename.
   - If you've created an HTML file to use as the alternate file, select Use Existing, then click Browse and select the HTML file.

6 Click OK to return to the Publish Settings dialog box.

Specifying publish settings for GIF files

GIF files provide an easy way to export drawings and simple animations for use in web pages. Standard GIF files are simply compressed bitmaps.

An animated GIF file (sometimes referred to as a GIF89a) offers a simple way to export short animation sequences. Flash optimizes an animated GIF file, storing only frame-to-frame changes.

Flash exports the first frame in the SWF file as a GIF file, unless you mark a different keyframe for export by entering the frame label #Static in the Property inspector. Flash exports all the frames in the current SWF file to an animated GIF file unless you specify a range of frames for export by entering the frame labels #First and #Last in the appropriate keyframes.

Flash can generate an image map for a GIF file to maintain URL links for buttons in the original document. You use the Property inspector to place the frame label #Map in the keyframe in which you want to create the image map. If you don't create a frame label, Flash creates an image map using the buttons in the last frame of the SWF file. You can create an image map only if the `$IM` template variable is present in the template you select. See "Creating an image map" on page 299.

To publish a GIF file with your Flash file:

1 Do one of the following to open the Publish Settings dialog box:

- Select File > Publish Settings.
- In the Property inspector for the document (which is available when no object is selected), click the Settings button.

Note: To create a publish profile for the publish settings that you'll specify, see "Using publish profiles" on page 295.

2 On the Formats tab, select the GIF Image type. In the File text box for the GIF image, either use the default filename, or enter a new filename with the .gif extension.

3 Click the GIF tab to display the file settings.

4 For Dimensions, enter a width and height in pixels for the exported bitmap image, or select Match Movie to make the GIF the same size as the Flash SWF file and maintain the aspect ratio of your original image.

5 Select a Playback option to determine whether Flash creates a still (Static) image or an animated GIF (Animation). If you select Animation, select Loop Continuously or enter the number of repetitions.

6 Select an option to specify a range of appearance settings for the exported GIF file:

Optimize Colors removes any unused colors from a GIF file's color table. This option reduces the file size by 1000 to 1500 bytes without affecting image quality, but slightly increases the memory requirements. This option has no effect on an adaptive palette. (An adaptive palette analyzes the colors in the image and creates a unique color table for the selected GIF file.)

Interlace incrementally displays the exported GIF file in a browser as it downloads. Interlacing lets the user see basic graphic content before the file has completely downloaded and may download the file faster over a slow network connection. Do not interlace an animated GIF image.

Smooth applies anti-aliasing to an exported bitmap to produce a higher-quality bitmap image and improve text display quality. However, smoothing may cause a halo of gray pixels to appear around an anti-aliased image placed on a colored background, and it increases the GIF file size. Export an image without smoothing if a halo appears or if you're placing a GIF transparency on a multicolored background.

Dither Solids applies dithering to solid colors as well as gradients. See Dither options in step 8.

Remove Gradients, turned off by default, converts all gradient fills in the SWF file to solid colors using the first color in the gradient. Gradients increase the size of a GIF file and often are of poor quality. If you use this option, select the first color of your gradients carefully to prevent unexpected results.

7 Select a Transparent option to determine the transparency of the application's background and the way alpha settings are converted to GIF:

Opaque makes the background a solid color.

Transparent makes the background transparent.

Alpha sets partial transparency. You can enter a Threshold value between 0 and 255. A lower value results in greater transparency. A value of 128 corresponds to 50% transparency.

8 Select a Dither option to specify how pixels of available colors are combined to simulate colors not available in the current palette. Dithering can improve color quality, but it increases the file size. Select from the following options:

None turns off dithering and replaces colors not in the basic color table with the solid color from the table that most closely approximates the specified color. Turning dithering off can result in smaller files but unsatisfactory colors.

Ordered provides good-quality dithering with the smallest increase in file size.

Diffusion provides the best-quality dithering but increases file size and processing time. It also works only with the web 216 color palette selected.

9 Select a Palette Type to define the image's color palette:

Web 216 uses the standard 216-color, browser-safe palette to create the GIF image, for good image quality and the fastest processing on the server.

Adaptive analyzes the colors in the image and creates a unique color table for the selected GIF file. This option is best for systems displaying thousands or millions of colors; it creates the most accurate color for the image but increases file size. To reduce the size of a GIF with an adaptive palette, use the Max Colors option in step 10 to decrease the number of colors in the palette.

Web Snap Adaptive is the same as the Adaptive palette option except that it converts very similar colors to the web 216 color palette. The resulting color palette is optimized for the image, but when possible, Flash uses colors from the web 216 palette. This produces better colors for the image when the web 216 palette is active on a 256-color system.

Custom specifies a palette that you have optimized for the selected image. The custom palette is processed at the same speed as the web 216 palette. To use this option, you should know how to create and use custom palettes. To select a custom palette, click the Ellipsis (...) button to the right of the Palette box at the bottom of the dialog box and select a palette file. Flash supports palettes saved in the ACT format, exported by Macromedia Fireworks and other leading graphics applications; for more information, see "Importing and exporting color palettes" on page 75.

10 If you selected the Adaptive or Web Snap Adaptive palette in step 9, enter a value for Max Colors to set the number of colors used in the GIF image. Choosing a smaller number of colors can produce a smaller file but may degrade the colors in the image.

11 Click OK to save the settings with the current file.

Specifying publish settings for JPEG files

The JPEG format lets you save an image as a highly compressed, 24-bit bitmap. Generally, GIF format is better for exporting line art, and JPEG format is better for images with continuous tones, such as photographs, gradients, or embedded bitmaps.

Flash exports the first frame in the SWF file as a JPEG, unless you mark a different keyframe for export by entering the frame label #Static.

To publish a JPEG file with your Flash SWF file:

1 Do one of the following to open the Publish Settings dialog box:
   - Select File > Publish Settings.
   - In the Property inspector for the document (which is available when no object is selected), click the Settings button.

   Note: To create a publish profile for the publish settings that you'll specify, see "Using publish profiles" on page 295.

2 On the Formats tab, select the JPEG Image type. For the JPEG filename, either use the default filename, or enter a new filename with the .jpg extension.

3 Click the JPEG panel to display its settings.

4 For Dimensions, enter a width and height in pixels for the exported bitmap image, or select Match Movie to make the JPEG image the same size as the Stage and maintain the aspect ratio of your original image.

5 For Quality, drag the slider or enter a value to control the amount of JPEG file compression used.

   The lower the image quality, the smaller the file, and vice versa. Try different settings to determine the best trade-off between size and quality.

   Note: You can set the bitmap export quality per object using the Bitmap Properties dialog box to change the object's compression setting. Selecting the default compression option in the Bitmap Properties dialog box applies the Publish Settings JPEG Quality option. See "Setting bitmap properties" on page 128.

6 Select Progressive to display Progressive JPEG images incrementally in a web browser, to make images appear faster when loaded over a slow network connection.

   This option is similar to interlacing in GIF and PNG images.

7 To save the settings with the current file, click OK.

Specifying publish settings for PNG files

PNG is the only cross-platform bitmap format that supports transparency (an alpha channel). It is also the native file format for Macromedia Fireworks.

Flash exports the first frame in the SWF file as a PNG file, unless you mark a different keyframe for export by entering the frame label #Static.

To publish a PNG file with your Flash SWF file:

1 Do one of the following to open the Publish Settings dialog box:
   - Select File > Publish Settings.
   - In the Property inspector for the document (which is available when no object is selected), click the Settings button.

   Note: To create a publish profile for the publish settings that you'll specify, see "Using publish profiles" on page 295.

2 On the Formats tab, select the PNG Image type. For the PNG filename, either use the default filename, or enter a new filename with the .png extension.

3 Click the PNG tab. For Dimensions, enter values for width and height in pixels for the exported bitmap image, or select Match Movie to make the PNG image the same size as the Flash SWF file and maintain the aspect ratio of your original image.